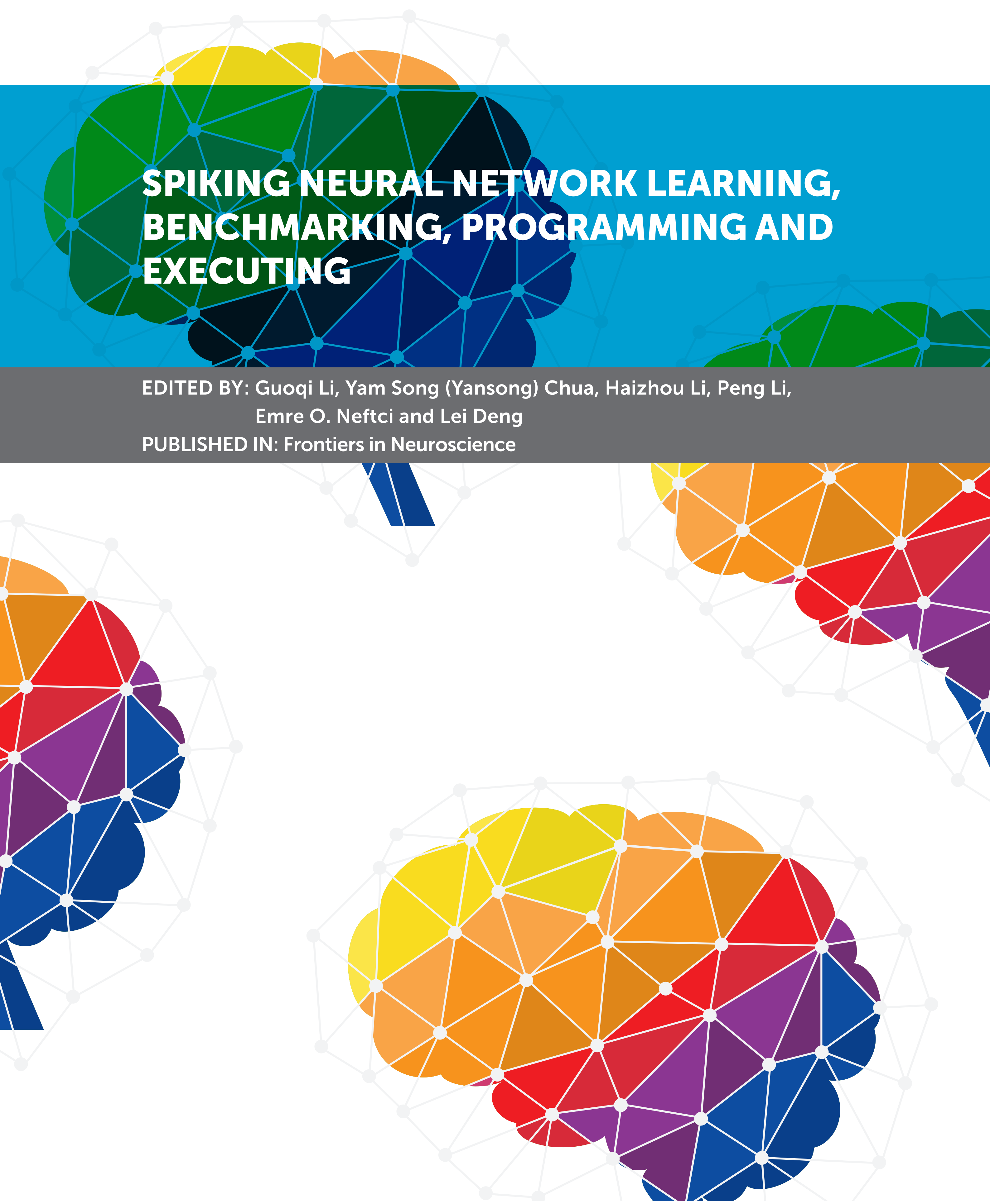
SPIKING NEURAL NETWORK LEARNING, BENCHMARKING, PROGRAMMING AND EXECUTING

EDITED BY: Guoqi Li, Yam Song (Yansong) Chua, Haizhou Li, Peng Li, Emre O. Neftci and Lei Deng

PUBLISHED IN: Frontiers in Neuroscience



frontiers Research Topics

ISSN 1664-8714
ISBN 978-2-88963-767-6
DOI 10.3389/978-2-88963-767-6

SPIKING NEURAL NETWORK LEARNING, BENCHMARKING, PROGRAMMING AND EXECUTING

Topic Editors:
Guoqi Li, Tsinghua University, China
Yam Song (Yansong) Chua, Huawei Technologies (China), China
Haizhou Li, National University of Singapore, Singapore
Peng Li, University of California, Santa Barbara, United States
Emre O. Neftci, University of California, Irvine, United States
Lei Deng, University of California, Santa Barbara, United States

Citation: Li, G., Chua, Y. S., Li, H., Li, P., Neftci, E. O., Deng, L., eds. (2020). Spiking Neural Network Learning, Benchmarking, Programming and Executing. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-88963-767-6

Table of Contents

EDITORIAL
published: 15 April 2020
doi: 10.3389/fnins.2020.00276

Editorial: Spiking Neural Network Learning, Benchmarking, Programming and Executing

Guoqi Li[1,2*†], *Lei Deng*[3†], *Yansong Chua*[4], *Peng Li*[3], *Emre O. Neftci*[5] *and Haizhou Li*[6]

[1] Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing, China, [2] Beijing Innovation Center for Future Chips, Tsinghua University, Beijing, China, [3] Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States, [4] Huawei Technologies, Shenzhen, China, [5] Department of Cognitive Science, University of California, Irvine, Irvine, CA, United States, [6] Department of Electrical Engineering, National University of Singapore, Singapore, Singapore

Keywords: deep spiking neural networks, SNN learning algorithms, programming framework, SNN benchmarks, neuromorphics

Editorial on the Research Topic

Spiking Neural Network Learning, Benchmarking, Programming and Executing

OPEN ACCESS

Edited by:
Timothy K. Horiuchi,
University of Maryland, United States

Reviewed by:
Scott Michael Koziol,
Baylor University, United States

Correspondence:
Guoqi Li
liguoqi@mail.tsinghua.edu.cn

†These authors have contributed equally to this work

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *13 November 2019*
Accepted: *10 March 2020*
Published: *15 April 2020*

Citation:
Li G, Deng L, Chua Y, Li P, Neftci EO and Li H (2020) Editorial: Spiking Neural Network Learning, Benchmarking, Programming and Executing. Front. Neurosci. 14:276. doi: 10.3389/fnins.2020.00276

INTRODUCTION

A spiking neural network (SNN), a type of brain-inspired neural network, mimics the biological brain, specifically, its neural codes, neuro-dynamics, and circuitry. SNNs have garnered great interest in both Artificial Intelligence (AI) and neuroscience communities given its great potential in biologically realistic modeling of human cognition and development of energy efficient, event-driven machine learning hardware (Pei et al., 2019; Roy et al., 2019). Significant progress has been made across a wide spectrum of AI fields, such as image processing, speech recognition, and machine translation. They are largely driven by the advance in Artificial Neural Networks (ANN) in systematic learning theories, explicit benchmarks with various tasks and data sets, friendly programming tools [e.g., TensorFlow (Abadi et al., 2016) and Pytorch (Paszke et al., 2019) machine learning tools], and efficient processing platforms [e.g., graphics processing unit (GPU) and tensor processing unit (TPU) (Jouppi et al., 2017)]. In comparison, SNNs are still at an early stage in these aspects. To further exploit the advantages of SNNs and attract more researchers to contribute in this field, we proposed a Research Topic in Frontiers in Neuroscience to discuss the main challenges and future prospects of SNNs, emphasizing on its "Learning algorithms, Benchmarking, Programming, and Executing." We are confident that SNNs will play a critical role in the development of energy efficient machine learning devices through algorithm-hardware co-design.

This Research Topic brings together researchers of different disciplines in order to present their recent work in SNNs. We received 22 submissions worldwide and accepted 15 papers. The scope of the accepted papers covers learning algorithms, model efficiency, programming tools, and neuromorphic hardware.

LEARNING ALGORITHMS

Learning algorithms play perhaps the most important role in AI techniques. Machine learning algorithms, in particular those for deep neural networks (DNN), have become the standard bearer in a wide spectrum of AI tasks. Some of the more common learning algorithms include backpropagation (Hecht-Nielsen, 1992), stochastic gradient descent (SGD) (Bottou, 2012), and

ADAM optimization (Kingma and Ba, 2014). Other techniques such as batch normalization (Ioffe and Szegedy, 2015) and distributed training (Dean et al., 2012) facilitate learning in DNNs and enable them to be applied in various real-world applications. In comparison, there are relatively fewer SNN learning algorithms and techniques. Existing SNN learning algorithms fall into three categories: unsupervised learning algorithms such as the original spike timing dependent plasticity (STDP) (Querlioz et al., 2013; Diehl and Cook, 2015; Kheradpisheh et al., 2016), indirect supervised learning such as ANN-to-SNN conversion (O'Connor et al., 2013; Pérez-Carrasco et al., 2013; Diehl et al., 2015; Sengupta et al., 2019), and direct supervised learning such as spatiotemporal backpropagation (Wu et al., 2018, 2019a,b). We note that progress in STDP research includes introducing a reward or supervision signal such as spike timing which, in combination with this third factor, dictates the weight changes (Paugam-Moisy et al., 2006; Franosch et al., 2013). Despite the progress made, no algorithm can yet train a very deep SNN efficiently, which has become almost the holy grail of our field. Below, we briefly summarize the accepted algorithm papers in this Research Topic.

Inspired by the mammalian olfactory system, Borthakur and Cleland develop an SNN model trained using STDP for signal restoration and identification. It is broadly applicable to sensor array inputs. Luo et al. propose a new weight update mechanism that adjusts synaptic weights, leading to the first wrong output spike-timing to classify input spike trains with time-sensitive information accurately. He et al. divide the learning (weight training) process into two phases: the structure formation phase using Hebb's rule, and the parameter training phase using STDP and reinforcement learning, so as to form an SNN-based associative memory system. In contrary to just training synaptic weights, Wang et al. propose training both the synaptic weights and delays using gradient descent so as to achieve better performance. Based on a structurally fixed small SNN with sparse recurrent connections, Ponghiran et al. use Q-learning to train only its output layer so as to achieve human-level performance on complex reinforcement learning tasks such as Atari games. Their research demonstrates that a small random recurrent SNN is able to provide a computationally efficient alternative to state-of-art deep reinforcement learning networks with several layers of trainable parameters. The above works have made good progress toward better performing SNN learning algorithms. We believe that further progress will be made in this field in the future.

MODEL EFFICIENCY

In recent years, hardware oriented DNN compression techniques have been proposed that offer significant memory saving and hardware acceleration (Han et al., 2015a, 2016; Zhang et al., 2016; Huang et al., 2017; Aimar et al., 2018). At present, many compression techniques are proposed that provide a trade-off between processing efficiency and application accuracy (Han et al., 2015b; Novikov et al., 2015; Zhou et al., 2016). Such an approach has also caught on in the design of SNN accelerators (Deng et al., 2019), with the following contribution in this Research Topic. Afshar et al. investigate how a hardware-efficient variant of STDP may be used for event-based feature extraction. Using a rigorous testing framework, a range of spatio-temporal kernels with different surface decaying methods, decay functions, receptive field sizes, feature numbers, and backend classifiers are evaluated. This detailed investigation provides useful insight and heuristics with regards to the trade-off between performance and complexity while using the STDP rule. Pedroni et al. study the impact of different arrangements of synaptic connectivity tables on weight storage and STDP updates for large-scale neuromorphic systems. Based on their analysis, they present an alternative formulation of STDP via a delayed causal update mechanism that permits efficient weight storage and access for both full and sparse connectivity.

Other than model complexity, several other papers focus on direct compression of SNNs. Soures and Kudithipudi propose Deep-LSM, a combination of randomly connected hidden layers and unsupervised winner-take-all layers to capture network dynamics followed by an attention modulated readout layer for classification. The connections between hidden layers and winner-take-all layers are partially trained using STDP. Their SNN model is applied in a first-person video activity recognition task, achieving state-of-the-art performance with $>90\%$ memory and operation saving compared to the long-short term memory (LSTM). Based on a single fully-connected layer with the STDP learning rule, Shi et al. propose a soft-pruning method that sets a fraction of the weights to the lower bound during training, effectively achieving $>75\%$ pruning. Srinivasan and Roy implement spiking convolutional layers comprising of binary weight kernels which are trained using probabilistic STDP, as well as non-spiking fully-connected layers which are trained using gradient descent. A residual convolutional SNN is proposed, which achieves >20x model compression.

PROGRAMMING TOOLS

Programming Tools have been one of the key components driving development in ANN research, examples of which include Theano (Al-Rfou et al., 2016), TensorFlow (Abadi et al., 2016), Caffe (Jia et al., 2014) and Pytorch (Paszke et al., 2019), MXNet (Chen et al., 2015), Keras (Chollet, 2015). These user-friendly programming tools enable researchers to build and train large-scale ANNs using only basic programming know-how. In comparison, the programming tools for SNNs are rather limited. To the best of our knowledge, only SpiNNaker (Furber et al., 2014), BindsNET (Hazan et al., 2018), and PyNN (Davison et al., 2009) provide a basic programming interface to support simple and small SNN simulations. Generally researchers have to build an SNN from the ground up which can be time-consuming and require significantly more programming know-how. Thus, developing user-friendly programming tools to efficiently deploy a large scale SNN is imperative to the advancement of our field. In this Research Topic, an open-source high-speed SNN simulation framework based on PyTorch has been proposed. SpykeTorch (Mozafari et al.) simulates convolutional SNNs with at most one spike per neuron (rank-order coding scheme), and STDP-based learning rules. Although programming tools for SNNs are still in their infancy, we believe that more research needs to be done so that the training of SNNs may approach the efficiency of training ANNs.

NEUROMORPHIC HARDWARES

Recent advances made in modeling SNNs *in-silico*, as demonstrated by Neurogrid of Stanford University (Benjamin et al., 2014), BrainScales of Heidelberg University (Schemmel et al., 2012), SpiNNaker of University of Manchester, Tianjic of Tsinghua University (Pei et al., 2019), IBM's TrueNorth (Akopyan et al., 2015), and Intel's Loihi (Davies et al., 2018), attest to the great potential of hardware implementation of SNNs. In this Research Topic, Shukla et al. re-model large-scale CNNs so as to mitigate hardware constraints and implement them on the IBM TrueNorth. A CNN used for car detection and counting was demonstrated, with reasonable accuracy compared to a GPU trained CNN but with much lower energy consumption. Bohnstingl et al. implement a learning-to-learn SNN on a neuromorphic chip which accelerates the learning process by extracting abstract knowledge from prior experiences. Other than conventional CMOS circuits, emerging devices such as memristors have also been studied in this Research Topic. Guo et al. propose a STDP-based greedy training algorithm for SNNs to reduce weight levels and enhance robustness toward device non-idealities. They demonstrate online learning on a resistive random access memory (RRAM) system with non-ideal behaviors. Fang et al. propose a generalized swarm intelligence model on SNN: the SI-SNN. SNNs are implemented as agents in swarm intelligence with interactive modulation and synchronization. They implement such neural dynamics on a ferroelectric field-effect transistor (FeFET) based hardware platform to solve optimization problems with high performance and efficiency.

CONCLUSIONS

In conclusion, it is believed that SNNs achieve superior performance in processing complex, sparse, and noisy spatiotemporal information with high power efficiency exploiting neural dynamics in the event-driven regime. Event-driven communication is particularly attractive in enabling energy efficient AI systems with in-memory computing that will play an important role in ubiquitous intelligence. SNN research is ongoing, and much more progress is to be expected in its learning algorithms, benchmarking framework, programming tools, and efficient hardware. Through cross-discipline exchange of ideas and collaborative research, we hope to build a truly energy-efficient and intelligent machine. This Research Topic is but a small step in this direction; we look forward to more disruptive ideas that distinguish SNNs and neuromorphic computing from the mainstream machine learning approaches in the near future.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

This work was partially supported by National Key R&D Program of China (No. 2018AAA0102600 and 2018YFE0200200), Beijing Academy of Artificial Intelligence (BAAI), Initiative Scientific Research Program, and a grant from the Institute for Guo Qiang, TsinghuaUniversity, and key scientific technological innovation research project by Ministry of Education, and Tsinghua–Foshan Innovation Special Fund.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). "Tensorflow: a system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (Savannah, GA), 265–283.

Aimar, A., Mostafa, H., Calabrese, E., Rios-Navarro, A., Tapiador-Morales, R., Lungu, I.-A., et al. (2018). Nullhop: a flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 644–656. doi: 10.1109/TNNLS.2018.2852335

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., et al. (2016). Theano: a Python framework for fast computation of mathematical expressions. *arXiv [preprint] arXiv*: 1605. 02688.

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J.-M., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Bottou, L. (2012). "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, eds G. Montavon, G. B. Orr, and K-R. Müller (Berlin; Heidelberg: Springer), 421–436.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., et al. (2015). Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv [preprint] arXiv*: 1512.01274.

Chollet, F. (2015). *Keras* [Online]. Available online at: https://keras.io.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.* 2:11. doi: 10.3389/neuro.11.011.2008

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems* (Lake Tahoe, NV), 1223–1231.

Deng, L., Wu, Y., Hu, Y., Liang, L., Li, G., Hu, X., et al. (2019). Comprehensive SNN compression using ADMM optimization and activity regularization. *arXiv [preprint] arXiv*: 1911.00822.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8.

Franosch, J. M. P., Urban, S., and van Hemmen, J. L. (2013). Supervised spike-timing-dependent plasticity: a spatiotemporal neuronal learning rule for function approximation and decisions. *Neural Comput.* 25, 3113–3130. doi: 10.1162/NECO_a_00520

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., et al. (2016). EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Comput. Architect. News* 44, 243–254. doi: 10.1145/3007787.3001163

Han, S., Mao, H., and Dally, W. J. (2015a). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv [preprint] arXiv*: 1510.00149.

Han, S., Pool, J., Tran, J., and Dally, W. (2015b)."Learning both weights and connections for efficient neural network,"in *Advances in Neural Information Processing Systems*, 1135–1143.

Hazan, H., Saunders, D. J., Khan, H., Patel, D., Sanghavi, D. T., Siegelmann, H. T., et al. (2018). Bindsnet: a machine learning-oriented spiking neural networks library in python. *Front. Neuroinform.* 12:89. doi: 10.3389/fninf.2018.00089

Hecht-Nielsen, R. (1992). "Theory of the backpropagation neural network,"" in *Neural Networks for Perception*, ed H. Wechsler (Elsevier), 65–93.

Huang, H., Ni, L., Wang, K., Wang, Y., and Yu, H. (2017). A highly parallel and energy efficient three-dimensional multilayer CMOS-RRAM accelerator for tensorized neural network. *IEEE Trans. Nanotechnol.* 17, 645–656. doi: 10.1109/TNANO.2017.2732698

Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv [preprint] arXiv*:1502.03167.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia* (Orlando, FL), 675–678.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON), 1–12.

Kheradpisheh, S. R., Ganjtabesh, M., and Masquelier, T. (2016). Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing* 205, 382–392. doi: 10.1016/j.neucom.2016.04.029

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv [preprint] arXiv*:1412.6980.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. (2015). "Tensorizing neural networks," in *Advances in Neural Information Processing Systems* (Montreal, QC), 442–450.

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.*7:178. doi: 10.3389/fnins.2013.00178

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). "PyTorch: an imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 8024–8035.

Paugam-Moisy, H., Martinez, R., and Bengio, S. (2006). *A supervised learning approach based on STDP and polychronization in spiking neuron networks*, IDIAP, EPFL.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Pérez-Carrasco, J. A., Zhao, B., Serrano, C., Acha, B., Serrano-Gotarredona, T., Chen, S., et al. (2013). Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 2706–2719. doi: 10.1109/TPAMI.2013.71

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans. Nanotechnol.* 12, 288–295. doi: 10.1109/TNANO.2013.2250995

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Schemmel, J., Grübl, A., Hartmann, S., Kononov, A., Mayr, C., Meier, K., et al. (2012). "Live demonstration: a scaled-down version of the brainscales wafer-scale neuromorphic system," in *2012 IEEE International Symposium on Circuits and Systems* (Seoul: IEEE), 702–702.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., and Li, H. (2019a). "Deep spiking neural network with spike count based learning rule," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest: IEEE), 1–6.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019b). "Direct training for spiking neural networks: faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Honolulu, HI), 1311–1318.

Zhang, S., Du, Z., Zhang, L., Lan, H., Liu, S., Li, L., et al. (2016). "Cambricon-x: an accelerator for sparse neural networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Taipei: IEEE), 1–12.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. (2016). Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv [preprint] arXiv*: 1606.06160.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 17 January 2019
doi: 10.3389/fnins.2018.01047

Investigation of Event-Based Surfaces for High-Speed Detection, Unsupervised Feature Extraction, and Object Recognition

Saeed Afshar, Tara Julia Hamilton, Jonathan Tapson, André van Schaik and Gregory Cohen*

Biomedical Engineering and Neuroscience Program, The MARCS Institute for Brain, Behaviour, and Development, Western Sydney University, Sydney, NSW, Australia

OPEN ACCESS

Edited by:
Haizhou Li,
National University of Singapore, Singapore

Reviewed by:
Huajin Tang,
Sichuan University, China
Arren Glover,
Fondazione Istituto Italiano di Technologia, Italy

****Correspondence:***
Saeed Afshar
s.afshar@westernsydney.edu.au

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *17 August 2018*
Accepted: *24 December 2018*
Published: *17 January 2019*

Citation:
Afshar S, Hamilton TJ, Tapson J, van Schaik A and Cohen G (2019) Investigation of Event-Based Surfaces for High-Speed Detection, Unsupervised Feature Extraction, and Object Recognition. Front. Neurosci. 12:1047. doi: 10.3389/fnins.2018.01047

In this work, we investigate event-based feature extraction through a rigorous framework of testing. We test a hardware efficient variant of Spike Timing Dependent Plasticity (STDP) on a range of spatio-temporal kernels with different surface decaying methods, decay functions, receptive field sizes, feature numbers, and back end classifiers. This detailed investigation can provide helpful insights and rules of thumb for performance vs. complexity trade-offs in more generalized networks, especially in the context of hardware implementation, where design choices can incur significant resource costs. The investigation is performed using a new dataset consisting of model airplanes being dropped free-hand close to the sensor. The target objects exhibit a wide range of relative orientations and velocities. This range of target velocities, analyzed in multiple configurations, allows a rigorous comparison of time-based decaying surfaces (time surfaces) vs. event index-based decaying surface (index surfaces), which are used to perform unsupervised feature extraction, followed by target detection and recognition. We examine each processing stage by comparison to the use of raw events, as well as a range of alternative layer structures, and the use of random features. By comparing results from a linear classifier and an ELM classifier, we evaluate how each element of the system affects accuracy. To generate time and index surfaces, the most commonly used kernels, namely event binning kernels, linearly, and exponentially decaying kernels, are investigated. Index surfaces were found to outperform time surfaces in recognition when invariance to target velocity was made a requirement. In the investigation of network structure, larger networks of neurons with large receptive field sizes were found to perform best. We find that a small number of event-based feature extractors can project the complex spatio-temporal event patterns of the dataset to an almost linearly separable representation in feature space, with best performing linear classifier achieving 98.75% recognition accuracy, using only 25 feature extracting neurons.

Keywords: event-based vision, recognition and classification, neuromorphic, event-based, unsupervided learning

INTRODUCTION

The last decade has seen significant development in the field of event-based cameras. Cameras such as the Dynamic Vision Sensor (DVS) (Lichtsteiner et al., 2008) and the Asynchronous Time-based Image Sensor (ATIS) (Posch et al., 2011) attempt to model the operation of the human retina by generating events at each pixel in response to changes in illumination. By only reporting changes in the visual field, event-based sensors perform compressive sensing at the pixel level, significantly reducing the output data-rate of the sensor relative to frame-based sensors that generate output regardless of the salience of its visual content. These cameras have spurred the development of a range of visual processing algorithms to tackle existing problems such as optical flow detection (Benosman et al., 2012), scene stitching (Klein et al., 2015), motion analysis (Litzenberger and Sabo, 2012), hand gesture recognition (Lee et al., 2014), hierarchical feature recognition (Orchard et al., 2015b), unsupervised visual feature extraction, and learning (Giulioni et al., 2015; Lagorce et al., 2015a), and tracking (Lagorce et al., 2015b; Glover and Bartolozzi, 2016, 2017). In addition to these works, in Ghosh et al. (2014) a frame based convolutional neural network was mapped to an event-based network using conversion of the event stream to static images via recent event presence, event counts, and event polarity. In Zhao et al. (2015), a hierarchical feature extractor network is presented where manually designed features are based on models of features in the visual cortex. In Peng et al. (2017), a bag of events method is used to perform feature extraction. An especially useful feature of this method is that only a single hyper-parameter needs to be tuned. This is in contrast to most proposed methods, which often have a large number of parameters, such that a rigorous analysis of their performance requires careful characterization and/or adversarial parameter selection, both of which are performed in this work.

More recently, the Hierarchy of Time Surfaces (HOTS) (Lagorce et al., 2017) was introduced which makes use of layers of time-decaying event-surfaces, or time surfaces, and feature-based clustering, with the features learnt in an unsupervised manner. The HOTS approach processes events in the temporal domain and is functionally similar to the feature extraction layer used in this work. The time surfaces which are used in HOTS and which also form part of the investigation in this work are a particularly effective method of implementing event-based convolutional networks.

In this work, we set out to rigorously quantify in detail the share in performance improvement attributable to each element of the system, namely: the memory generation and decay methods, commonly used memory kernels, use of raw events relative to use of feature events, the event-based convolutional structure of the feature extractors and the performance of the back-end classifier.

An important question arising at every stage of any event-based algorithm is whether the event rate should inform the progression of the algorithm through time. In this work, we investigate this question through comparisons of time surfaces and index surfaces where the memory of events decay as a function of time or event index, respectively.

Processing event memory as a function of time is straight-forward and intuitive. By decaying event memory as a function of time, all elements of an event-based system operate in a uniform time-based manner regardless of the informational content in any part of the sensor's field of view. The behavior of time-based decaying memory does not vary as a function of sensor size or any aspect of the visual scene that alters the event generation rate, such as scene contrast or texture. However, once the sensor event rate is incorporated into the operation of the system, these invariances may no longer hold, since a change in event rate may alter the decay rate of the memory of the event stream, potentially resulting in information loss. Therefore, algorithms using event rate information in memory decay require more careful testing, parameter selection, and potentially secondary solutions such as localized memory decay mechanisms to mitigate information loss. On the other hand, processing event memory as a function event count or index does have one crucial advantage over a purely time-based processing system. In general, event-based vision sensors generate more events in response to faster moving objects when holding other variables constant. This approximately proportional relationship between local event rate and local velocity allows an algorithm operating as a function of event index to effectively make computational decisions at approximately the same speed as the object being observed. Previous works have suggested that the use of event index to decay memory provides greater robustness in the presence of such variance in target velocity (Ghosh et al., 2014; Glover and Bartolozzi, 2016, 2017). In Glover and Bartolozzi (2016) an event-based Hough transform was used for tracking and in Ghosh et al. (2014) this was augmented with an event-based particle filter to improve tracking performance. The Hough transform in these works was implemented using a window of fixed event size, thus incorporating the event-rate information into the algorithm. The results showed that higher target velocities increased the update rate of the algorithm, allowing better tracking performance at high velocity. In Ghosh et al. (2014), windows of fixed event number and fixed time windows were compared in their performance in simultaneous tracking and recognition, and a slightly higher recognition accuracy was achieved when the algorithm was tested for velocity invariance. Such robustness to observed velocities in the data can be critical in a range of real world applications. These results, and the potential utility of velocity robust algorithms in real world applications of event-based sensors, motivate a central element of the investigation presented in this work. One such example is one of the few current applications of event-based sensors: the field of event-based Space Situational Awareness (SSA), where event-based sensors uniquely allow observation and tracking of non-terrestrial targets during both night and day (Cohen et al., 2017). However, a major challenge in such a task is the extremely limited collection of event-based observations of objects of interest. A major aspect of this limitation is that particular targets may only have been observed at a single velocity relative to the sensor yet must be detected, tracked, and identified robustly regardless of their relative velocity. This requirement of robustness to target velocity variations motivates the detailed rigorous examination of time and index surfaces in combination with a range of commonly used decay kernels.

Another important element in a wide range of event-based algorithms is the use of feature extractors. The contribution of the feature extraction layer as a whole is the simplest to determine and yet can often be missing in the literature as a baseline performance measure. This involves directly feeding sensor events into the final stage classifiers in the same manner as the output feature layer, skipping the intervening feature extraction layer(s). A more subtle question is how effective the learnt features are. In other words, how well does the learning algorithm orient the feature set with respect to the data so as to cover the underlying non-linearities in the dataset? This can be ascertained by comparing the mean recognition performance of multiple independently learnt features against random instantiations of features with the same network structure and feature weight distribution. The power of random features to cover non-linear feature spaces has been demonstrated by the Extreme Learning Machine (Clady et al., 2015) literature. By comparing feature extraction algorithms to a baseline of random features a better understanding of the relative improvement can be ascertained.

Finally, the most complex measure that is investigated is the role of the classifier on the performance. While there are a wide range of potential back-end classifiers that may be used, we propose that the combined use of linear classifiers and large hidden layer ELMs have particular utility in providing a rigorous measure of residual non-linearity following each stage of processing. This is because, unlike other classifiers, which through learning orient their non-linear features toward the training data, the random non-linear projections of the ELM's hidden layer create projections that are approximately uniform with regard to the structure of the data. As such the size of the hidden layer provides a reasonably "unbiased" measure of the residual non-linearities present after each a processing layer.

METHODOLOGY

Generating the Dataset

The system presented in this paper constitutes an event-based and high-speed classification system, and makes use of a real-world task, and its associated dataset, to demonstrate and characterize its performance.

A variety of event-based datasets now exist, such as the N-MNIST and N-Caltech101 (Orchard et al., 2015a), MNIST_DVS (Serrano-Gotarredona and Linares-Barranco, 2015), and the event-based UCF-50 datasets (Hu et al., 2016). One common facet of these datasets is that they have been generated under highly constrained conditions, especially with respect to the range of target object velocities. For a static image, event-based cameras only produce data in response to motion and therefore require either the static image, or the camera itself to be moving. Therefore, the velocities involved in many of the event-based datasets are strictly controlled. This is often a desirable trait to ensure consistency across all samples, but this constraint is a strongly artificial one. Other event-based datasets, such as the visual navigation dataset found in Barranco et al. (2016), do not control velocity in the same manner, but represent a fundamentally different task and are therefore not well-suited to exploring detection and feature extraction mechanisms.

The need to explore the effect of variances in velocity is important as these tend to produce significant variance in the spatio-temporal event patterns generated by event-based cameras. This can have a significant impact on the performance of a classifier or detection algorithm. A primary focus of this work is on the comparison of different event-based processing approaches in the presence of such variance. This required the creation of a new dataset designed to test event-based classification algorithms under conditions that are less constrained and closer to those found in real-world tasks. However, as well as being reasonably difficult, the dataset was

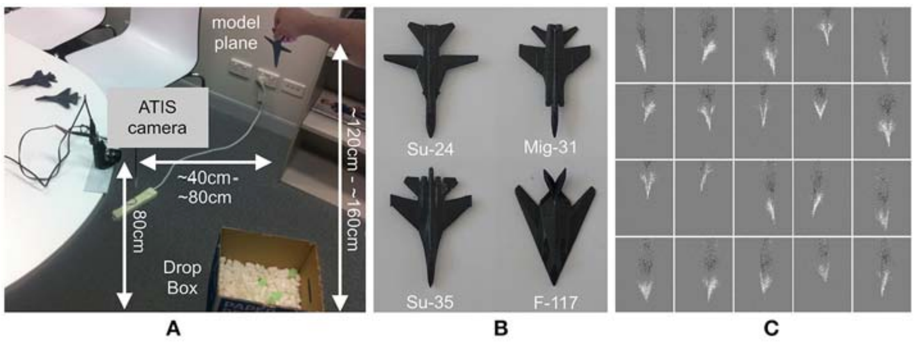


FIGURE 1 | Data collection setup and samples of the airplane dropping dataset. **(A)** The physical setup used for recording dataset in which an ATIS camera is attached to a table and the airplanes dropped freehand in front of the camera. **(B)** A top-down and labeled view of the four model airplanes used to generate the dataset. **(C)** Examples of the variation in the dataset in terms of position, scale, orientation, and speed. Each image represents a frame rendered from the same 3 ms of events extracted from each recording with ON events represented with white pixels and OFF events represented with black pixels. The twenty random samples clearly demonstrate the difficulty of the recognition task. Unlike most event-based datasets, the camera was not tuned or biased for the application, simulating real world noisy dynamic environments where such fine tuning would be difficult or impossible. As a result of this arbitrary untuned camera configuration the OFF events (black) in the entire dataset produced essentially noise clouds and as such were discarded. Airplane class key ordered from top left to bottom right, Mig-31: {2, 3, 7, 11, 12}, F-117: {9, 15, 16, 18, 19}, Su-24: {1, 5, 8, 14, 20}, and Su-35: {4, 6, 10, 13, 17}.

also designed to be constrained enough to allow a rigorous comparison of the various parameters and architectures of interest. As such the dataset was specifically designed to act as a proxy for a noisy local region in a larger real-world dataset.

The task is to identify model airplanes as they rapidly pass through the field of view of an ATIS camera. The airplanes were dropped free-hand, and from varying heights and distances from the camera, as shown in **Figure 1A**. Four model airplanes were used, each made from steel and all painted uniform gray, as shown in **Figure 1B**. This served to remove any distinctive textures or marking from the airplanes, thereby increasing the difficulty of the task. The airplanes are models of a Mig-31, an F-117, a Su-24, and a Su-35, with wingspans of 9.1, 7.5, 10.3, and 9.0 cm, respectively.

The recordings were captured using the same model of ATIS camera and the same acquisition software used in capturing the N-MNIST dataset in Orchard et al. (2015a), and the recordings were stored in the same file formats, thereby maximizing compatibility with other neuromorphic algorithms and systems. The models were dropped 100 times each from a distance ranging from 120 to 160 cm above the ground and at a horizontal distance of 40 to 80 cm from the camera. This ensured that the airplanes passed rapidly through the field of view of the camera, with the planes crossing the field of view in an average of 242 $\pm$ 21 ms. No mechanisms were used to enforce consistency of the airplane drops, resulting in a wide range of observed speeds from 0 to $>$1500 pixels per second. Additionally, there were variable delays before and after each drop, resulting in recordings of varying lengths. The dataset was augmented with left-right flipped versions of the recordings, resulting in 200 drops for each airplane type. An example of the variability in the airplane drops is demonstrated in **Figure 1C**, which shows binned events in the same 3 ms slice of data from 20 randomly selected recordings from the dataset. The samples demonstrate significant variations in the positions of the airplanes, their orientations, and their sizes. No attempt was made to fine tune the sensors biases for the particular light condition or target velocities. This lack of tuning is likely in real-world environments where the recording conditions may not be known a priori. An example of this is the previously mentioned SSA application (Cohen et al., 2017), where acquired data is inevitably noisy, often with one of sensors polarities entirely unable to capture useful events from the target due to the sensor biases not being matched to the lighting or velocity profile of the target. Even when the sensor biases are ideal for the lighting and temperature conditions of the recording, there are always fainter targets of interest in the field of view which can only be viewed by lowering sensor biases and "delving deeper into the noise" to accumulate events from these fainter objects. Thus, allowing noise and un-tuned biases into datasets, additional real-world challenges, such as structured noise and unevenly performing polarities, become apparent, motivating the implementation of robust solutions and new network behaviors that would otherwise be missed.

Figure 2A shows the event time vs. event index profiles of all recordings in the dataset showing the significant inter and intra recording variance in data-rate present in the dataset. While the number of recordings in the augmented dataset is 800, the number of surface samples making up the data points presented to the detection and recognition algorithm is $>$20,000 samples. The free-hand drop methodology resulted in significant variance in velocity and orientation of the model airplane within each recording. As a result, the spatio-temporal output

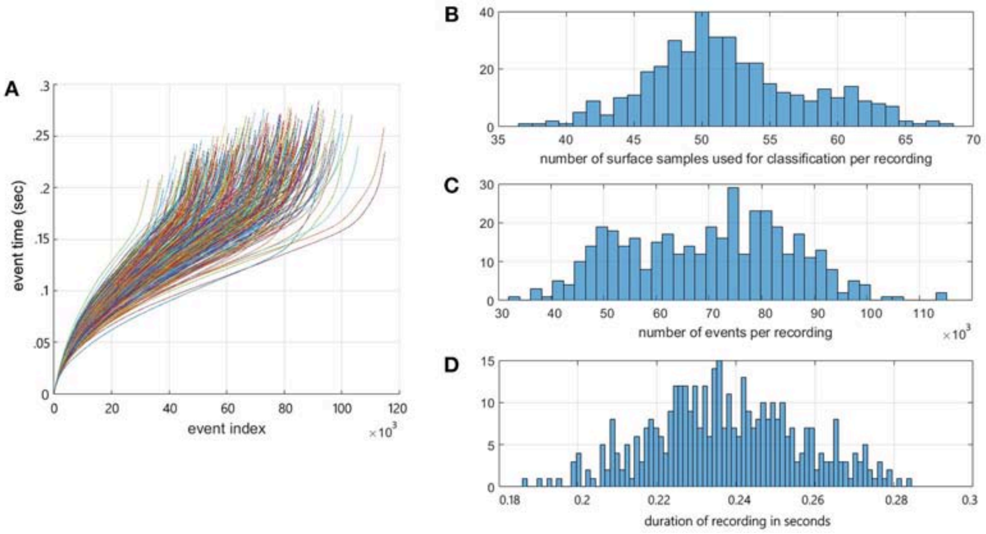


FIGURE 2 | The Dataset Summary. **(A)** Event timestamp profiles of all airplane drops in the dataset showing the event timestamps of each recording as a function of event index. The timestamp profiles demonstrate the variable rates of event generation within and across the recordings. These differences are a function of the speed, size, and shape of the airplanes and the distance from the camera. Note the color assigned to each recording profile is arbitrary. **(B)** Distribution of the number of frames per recording for each recording in the dataset. **(C)** Distribution of the number of events per recording for each recording in the dataset. **(D)** Distribution of the duration of each recording in the dataset.

patterns varied significantly through each recording, as shown in **Figure 2A** and discussed in later sections. The distribution of the number of surface samples per recording is shown in **Figure 2B**. **Figures 2C,D** show the distribution in the number of events per recording and recording duration for the dataset. The full dataset can be found at Afshar et al. (2018).

Time-Surface vs. Index Surfaces

An event ev_i from the ATIS camera can be described mathematically by:

$$ev_i = [\mathbf{x_i}, t_i, p_i]^T \tag{1}$$

where i is the index of the event, $\mathbf{x_i} = [x_i, y_i]$ is the spatial address of the source pixel corresponding to the physical location on the sensor, $p_i \in \{-1\ ,\ 1\}$ is the polarity of event indicating whether the log intensity increased or decreased, and t_i is the absolute time at which the event occurred (Clady et al., 2015). The time t_i is applied to the event by the ATIS camera hardware and has a resolution of 1 ms.

Event-based algorithms require iterative processing of each event, and therefore require that each new observation be combined with previously observed local events, both in space and in time. This is accomplished using a variation of the time surfaces from the HOTS algorithm (Lagorce et al., 2017), but extended to cover surfaces decaying based on time (time surface) and based on event index (index surface). Each new incoming event updates the surface and defines a region representing the spatio-temporal neighborhood on which further processing may be performed.

The timing and polarity information contained in each event, as shown in equation (1), allows the generation of two useful surfaces, based on time and polarity, from which more complex surfaces can be constructed. The first surface, referred to as T_i, maps the time of the most recent event to spatial pixel location and is described in (2), with the corresponding surface P_i for event polarity given by (3). Note as discussed above due to the noisiness of the OFF events due to untuned biases, only ON events with $p_i = 1$ were used.

$$\begin{aligned} T_i &: \mathrm{R}^2 \to \mathrm{R} \\ \mathbf{x} &: t \to\ T_i(\mathbf{x}) \end{aligned} \tag{2}$$

$$\begin{aligned} P_i &: \mathrm{R}^2 \to \{-1, 1\} \\ \mathbf{x} &: p \to P_i(\mathbf{x}) \end{aligned} \tag{3}$$

Here, we compare the time surfaces introduced in the HOTS algorithm, which decay as a function of time, with index surfaces, where the surface values for all pixels decay not as a function of time, but in response to new incoming events. We then define the analogous function to (2) for index surfaces. This surface, I_i, is defined in (4) and stores the indices of incoming event for each spatial pixel.

$$\begin{aligned} I_i &: \mathrm{R}^2 \to \mathrm{R} \\ \mathbf{x} &: i \to\ I_i(\mathbf{x}) \end{aligned} \tag{4}$$

In addition to exploring time-based decay and index-based decay, three different transfer functions or temporal kernels are investigated. These kernels are event binning (*BTS*/*BIS*), linear decay (*LTS*/*LIS*) and exponential decay (*ETS*/*EIS*). As a point of reference, the HOTS algorithm makes use of exponential decaying time kernels.

In all surface generation methods, when a new event arrives, the surface at $\mathbf{x_i}$ is set to P_i. When using the event binning technique, the value on the surface maintains its value over a temporal window τ_e or index window N_e, after which it is reset to zero. The event binning method for surface generation is described by equations (5) for the time-based binning (*BTS*) and (6) for the index-based binning (*BIS*).

$$BTS_i(\mathbf{x}, t) = \begin{cases} P_i(\mathbf{x}), & t - T_i(\mathbf{x}) \le \tau_e \\ 0, & t -\ T_i(\mathbf{x}) > \tau_e \end{cases} \tag{5}$$

$$BIS_i(\mathbf{x}) = \begin{cases} P_i(\mathbf{x}), & i - I_i(\mathbf{x}) \le N_e \\ 0, & i - I_i(\mathbf{x}) > N_e \end{cases} \tag{6}$$

For the linearly decaying time surface (*LTS*) and linearly decaying index surface (*LIS*), the initial value set on the surface in response to a new event instead decays toward zero linearly as a function of time. These surfaces are described by (7) for time-based linear decay or in response to incoming events as described by (8) for index-based linear decay.

$$LTS_i(\mathbf{x}, t) = \begin{cases} P_i(\mathbf{x}).(1 + \frac{T_i(\mathbf{x})-t}{2\tau_e}), & t{-}T_i(\mathbf{x}) \ge 2\tau_e \\ 0, & t{-}T_i(\mathbf{x}) < 2\tau_e \end{cases} \tag{7}$$

$$LIS_i(\mathbf{x}) = \begin{cases} P_i(\mathbf{x}).(1 + \frac{I_i(\mathbf{x})-i}{2N_e}), & i - I_i(\mathbf{x}) \ge 2N_e \\ 0, & i - I_i(\mathbf{x}) < 2N_e \end{cases} \tag{8}$$

The exponential decay method works in a similar manner to the linear decay, with the value placed on the surface decaying exponentially instead of linearly with respect to either time or event. This results in the equations for the exponentially decaying time surface (*ETS*) shown in (9), and the exponentially decaying index surface (*EIS*) shown in (10).

$$ETS_i(\mathbf{x}, t) = P_i(\mathbf{x}).e^{\frac{T_i(\mathbf{x})-t}{\tau_e}} \tag{9}$$

$$EIS_i(\mathbf{x}) = P_i(\mathbf{x}).e^{\frac{I_i(\mathbf{x})-i}{N_e}} \tag{10}$$

The equations for these surfaces make use of a constant parameter, time constant τ_e for time-based methods and index constant N_e for the index-based methods and the chosen values for these parameters are shown in **Figures 3A,B**. The plots show the time surface and index surface generation kernels which have an area under the curve of 3 ms in (a), and 554 events in (b), respectively. These values were chosen based on the mean data rate over all recordings.

Given the 184.5 k event/s event rate for the entire dataset the area under the curves in **Figures 3A,B**, $\tau_e = 3$ and $N_e = 554$, respectively were chosen to be approximately equal, thus resulting in approximately equal total surface activation for the time and index based decay methods over the entire dataset, but not for any individual recording or section thereof.

To illustrate the difference in the two decay methods, **Figure 4** shows index surface subtracted from the time surface for a

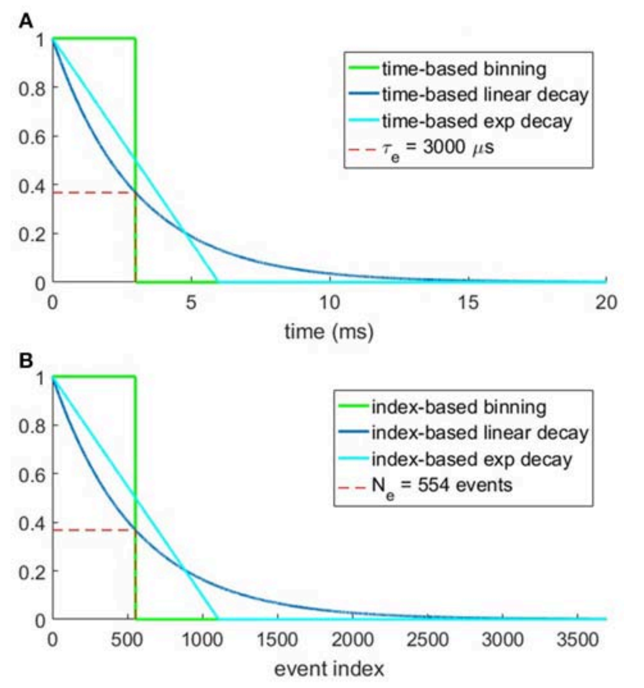


FIGURE 3 | Plots of the six methods for generating time and index surfaces. **(A)** Shows the three time-based kernels over time. Note that the area under all kernels is the time constant τ_e=3 ms. **(B)** shows the value of the index-based kernel as a function of event index. Here the mean dataset event rate over all recordings (~184.5 k events/s) was used to obtain equivalent sized kernels with index constant N_e=554 events.

single recording from the dataset. The figure shows that the binning time surface has a lower activation than the binning index surface when the speed of the airplane is low (at the start of the recording). As the airplane speeds up through its fall, the total time surface activation continues to increase whilst the index surface remains approximately constant. In fact, at t = 157 ms, the total activation on the time surface is approximately twice that of the index surface which remains relative stable throughout the recording. This stability of index surface activation is the direct result of the decay process. Since both the increase and decrease in surface activation are a function of event index, all decay kernels with a finite impulse response will inevitably generate stable surface activations. This is in contrast to the time decay method where no coupling exists between the activation and decay of the surface. **Figures 4D–F** show that the difference between the two decay methods are greatest for the binning method, followed by linear decay and finally exponential decay, which is the result of a slight reduction in surface activation from binning to linear to exponential decay for the time surfaces. This reduction is due to the kernel width such that the arrival of new events overwrite the entries for pixels that have recently been activated. This effect is more pronounced for kernels with a longer time window as the surface maintains the value for longer. This same effect is also present in the index surfaces, but is less prominent due to the lower variance of the index-based activation plots. Overall, **Figure 4** highlights the event-overwrite effect for different decay methods and kernels, as well as the significantly lower variance of index surface activation in the presence of change in velocity (due to gravity) relative to time surfaces. Such lower variance potentially allows downstream processing stages to be optimized for the stable operating point of the index surface.

Target Velocity vs. Surface Activation

Prior to the feature extraction and recognition, the airplane is detected and the location within the field of view is determined. The speed of the airplanes is much faster than any other stimulus expected within the field of view of the camera, such as the body of the author accidently entering the frame, as can be seen in the lower right pane of **Figure 5c**. Therefore, summation of events across the rows and columns of the camera's field of view (after normalization and thresholding as shown in **Figures 5a,b** provides a simple method to detect the boundary of the airplane in the limited context of this investigation. While the presence of slow moving objects in the background can be rejected as shown in **Figure 5c**, complex background objects with similar velocities to the target would impair this simple object detector.

In terms of limitations, the presented dataset is constrained in the sense of having only a single high-speed object in the field of view against an effectively blank background. This restriction allows a more focused investigation of different methodologies as well as of the sources of variance in the data such as target orientation and velocity. While the restriction may appear to limit the generalization of the results to more complex scenes, the dataset and the resulting network solutions should be viewed as investigating a local region within a more complex visual scene and the processing required for it which would be represent a small section in a larger system.

By using the detection method described we can plot the estimated vertical position of each target airplane as shown in **Figure 6**, both in terms of time in **Figure 6A** and event index **Figure 6B**. These vertical position profiles serve to further highlight the difference between the index-based and time-based approaches in the context of local velocity. Whereas, the estimated position plots take on their expected parabolic shape when plotted against time, when plotted against index, the trajectories are linear to a first approximation. The linearity of target position with respect to event index provides an interesting insight into the potential use of index surfaces for tracking, however, this is beyond the scope of the work presented here, which focuses on detection and recognition.

Figure 7 illustrates the wide range of velocities in the dataset and the associated mean rate of change in surface activation for time surfaces, index surfaces. The exponentially decay kernel was used for this test. The line of best fit through the data demonstrates different relationships between velocity and change in surface activation which arise from the different geometries of the airplanes. In all cases, however, surface activation is significantly more sensitive to velocity when using time surfaces than index surfaces. This invariance hints at potential utility of index surfaces for velocity invariant feature generation, where features learnt from a dataset with a particular velocity distribution operate equally well on a dataset with an entirely

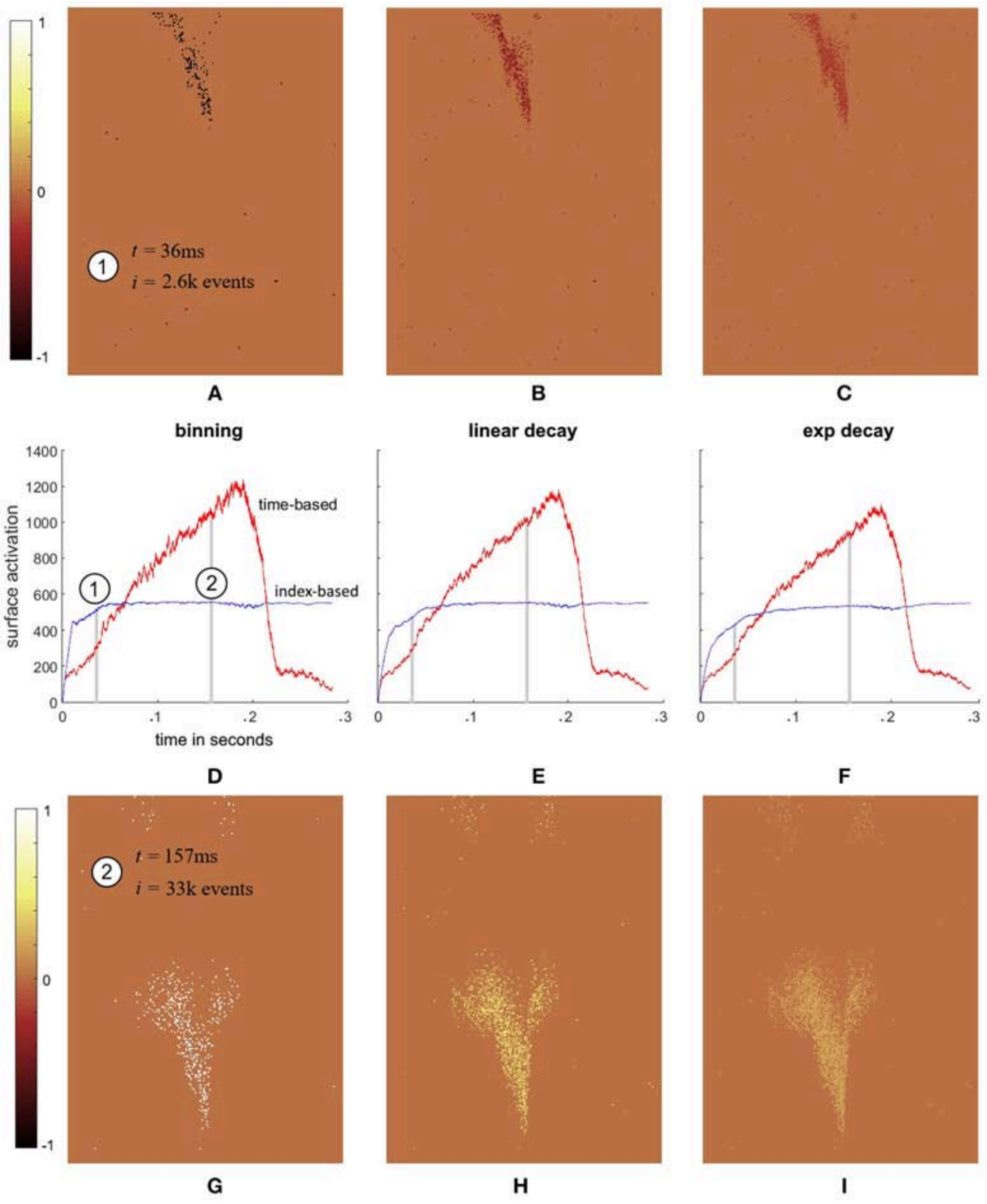


FIGURE 4 | Comparison of surface activation for a single recording. **(A–C)** show the surface differences ($BTS_i - BIS_i$), ($LTS_i - LIS_i$), and ($ETS_i - EIS_i$), respectively at the beginning of the recording (t=36 ms). This moment in the recording is marked (1) on **(D)** which displays total surface activation for the binning method $\sum_{x,y} BTS_i$ and $\sum_{x,y} BIS_i$. The two traces in **(D)** show that at the beginning of the recording when the target airplane's speed is low the binning time surface has a lower activation than the binning index surface. However, as the target speeds up, the total time surface activation also increases, while the index surface remains approximately stable, such that by t=157 ms the time surface activation $\sum_{x,y} BTS_i$ is approximately twice that of $\sum_{x,y} BIS_i$. **(E,F)** show a similar but slightly less pronounced relative increase for the linear and exponential decay surfaces. **(G–I)** show this relative increase for the binning, linear, and exponential decay surfaces by plotting the differences of **(A–C)** at t=157 ms.

different velocity distribution, which is not the case for time surfaces. We explore the ramifications of this invariance further in section Velocity Segregated Dataset.

Event-Based Feature Extraction

An event-based feature extractor was used to learn the most common spatio-temporal features generated by the recordings. The unsupervised spike-based feature extraction algorithm was developed for hardware implementation, as previously described in Afshar et al. (2014). In this algorithm, the Synapto-dendritic Kernel Adaptation Network (SKAN), a single layer of neurons with adaptive synaptic kernels and adaptive thresholds compete in the temporal domain to learn commonly observed spatio-temporal spike patterns. These adaptive synapto-dendritic

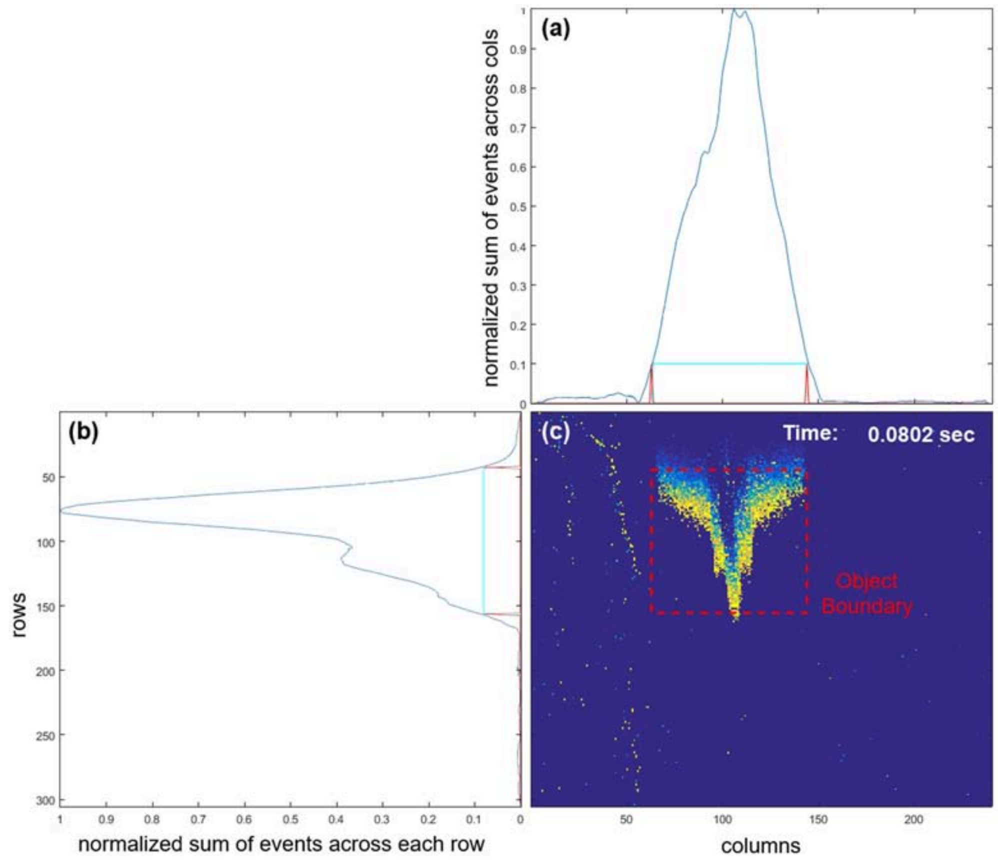


FIGURE 5 | Screenshot from a live demonstration of the airplane drop test after 0.08 s. **(a,b)** are a smoothed summation of recent events across columns and rows, respectively. The smoothing was performed by using an 8-pixel wide rectangular moving average window. Due to the relatively high speed of the airplane these summations, when normalized and thresholded at 0.1, could reliably be used to extract the fast-moving airplane from the static background or slower moving objects. The generated target object's boundary is shown in **(c)**. Note that movement of the body of the author (light vertical trace on the left) as he drops the airplane is slow relative to the airplane and generates relatively few events and so does not reach even the low set (th = 0.1) detection threshold.

kernels provide an abstracted representation of the coupling of pre- and post-synaptic neurons via multiple synaptic and dendritic pathways allowing unsupervised learning and inference of precise spike timings. By conceptually combining multiple synapses, the most numerous elements of any neuromorphic system, into a single adaptive kernel, the SKAN algorithm allows an efficient yet reasonably complex model of STDP to be realized in hardware. In Afshar et al. (2015) the algorithm was extended using a simplified model of Spike Timing Dependent Plasticity (STDP) (Markram et al., 1997) to provide synaptic encoding of afferent Signal to Noise Ratio. In Sofatzis et al. (2014) the algorithm was used to perform real-time unsupervised hand gesture recognition using an FPGA. In this work, the event-based approach is continued at the feature extraction layer with the output spike of the winning neuron representing a *feature event*.

The SKAN layer operates via two simple feedback loops: a synaptic kernel adaptation loop and a threshold adaptation loop. Each input event $u_i(t)$ in a spatio-temporal pattern activates a triangular post synaptic kernel $\mathrm{r}_i(\mathrm{t})$ as described by (11) and (12). The kernels are summed at the soma to generate a membrane potential. While this membrane potential is above the neurons adaptive threshold $\Theta(t)$, the neuron output $s(t)$ goes high, which is analogous to a series of action potentials or a neuronal burst, as described in (13). While the neuron output $s(t)$ is high, the kernels perform their temporal adaptation operation as described by (12). According to this rule every time step where the neuron output is high and the kernel is rising ($p_i = 1$), the synaptic kernel's slope Δr_i is reduced by a small amount ddr, thus moving the kernel peak later in time to better match the observed pattern. Conversely if the event is too early, the kernel's slope Δr_i is raised contracting the kernel and moving its peak earlier in time.

$$p_i(t) = \begin{cases} 1 \quad \text{if } \left(u_i(t) = 1 \wedge p_i(t-1) = 0\right) \\ \quad \vee \left(p_i(t-1) = 1 \wedge r_i(t-1) < w_i\right) \\ -1 \quad \text{if } \left(p_i(t-1) = 1 \wedge r_i(t-1) \geq w_i\right) \\ \quad \vee \left(p_i(t-1) = -1 \wedge r_i(t-1) > 0\right) \\ 0 \quad \text{else} \end{cases} \tag{11}$$

$$\begin{bmatrix} r_i(t) \\ \Delta r_i(t) \end{bmatrix} = \begin{bmatrix} r_i(t-1) \\ \Delta r_i(t-1) \end{bmatrix} + p_i(t-1) \begin{bmatrix} \Delta r_i(t-1) \\ ddr \times s(t-1) \end{bmatrix} \tag{12}$$

$$s(t) = \begin{cases} 1 \quad \text{if } \sum_i r_i(t) > \Theta(t-1) \\ 0 \quad \text{else} \end{cases} \tag{13}$$

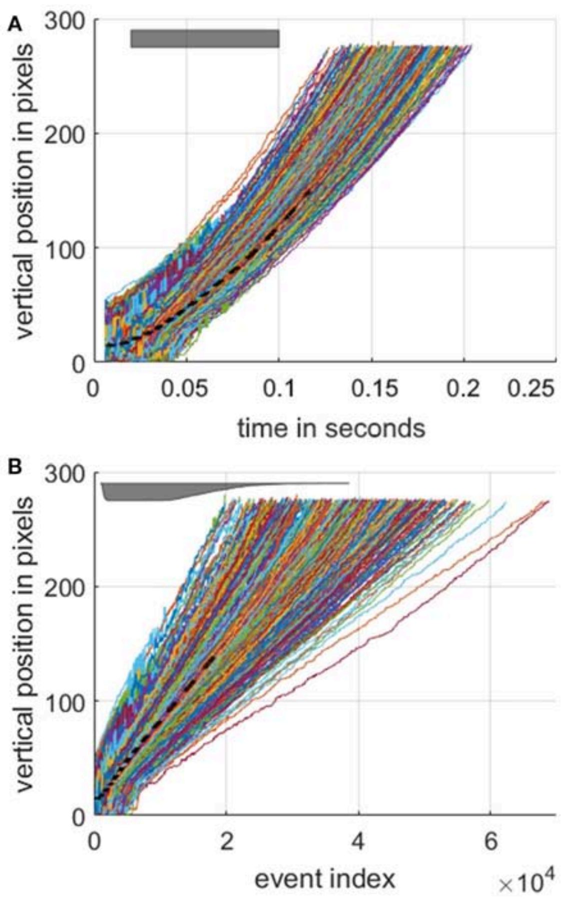


FIGURE 6 | Estimated vertical position of the target as a function of time **(A)** and as a function of event index **(B)**. The dashed black line marks the mean position over all recordings. For the entire dataset, the mean time interval from the first valid object boundary detection event to the last was 156.2 ms with a standard deviation of 17.8 ms. The target's position was defined as the midpoint between the object boundaries as shown in **Figure 5 (C)**. The gray bar at the top left in **(A)** indicates the time window used for investigating the effect of target velocities on surface activation in **Figure 7**. The same gray time window bar is shown in lower **(B)** panel as a function of event index. The relative thickness of the bar is proportional number of recordings in the time window of **(A)** at each event index. Note the color assigned to each recording profile is arbitrary.

The neuron's thresholds adapt via a similar mechanism to the kernels. At each time step where the neuron output is high the neuron's threshold also rises. In addition at the falling edge of the neuron output pulse, the threshold falls by a small value. A single inhibitory neuron prevents multiple neurons spiking at the same time thus preventing duplicate learning of the same pattern by multiple neurons.

$$\Theta(t) = \begin{cases} \Theta(t-1) + \Theta_{rise} & \text{if } \sum_i r_i(t) > \Theta(t-1) \\ \Theta(t-1) - \Theta_{fall} & \text{if } \sum_i r_i(t) \\ & \quad = 0 \wedge \sum_i r_i(t-1) > 0 \\ \Theta(t-1) & \text{else} \end{cases} \tag{14}$$

This simple hardware implementable rule-set allows the neurons to orient their spatio-temporal receptive fields from a random starting point toward the most commonly observed patterns, thus attempting to optimally represent the observed data given a limited number of features. It is in the class of unsupervised training algorithms used in wide range of neuromorphic algorithms such as STDP. For detailed description of the hardware implementation of the algorithm and resultant behaviors see (Afshar et al., 2014).

When the camera detects a new event, a 13 × 13-pixel region of the surface around it is converted to a temporally coded spatio-temporal spike pattern. This value to time encoding method was originally used in Masquelier and Thorpe (2007). The normalized real-valued intensity of the surface is first rescaled from 0–1 to 0–255 and then mapped to an 8-bit unsigned integer. This 8-bit encoding of the surface allows for potential hardware implementation of the SKAN kernels, without needing floating point operations. This integer representation of the local surface region is then encoded into spike delays forming a spatio-temporal spike pattern. The resultant pattern is then used as the input to a 25-neuron network. The neurons were trained 10 times independently using half the dataset consisting of 50 recordings from each plane type augmented by the left-right flipped version of these recordings. Learning (adaptation) in the feature detection neurons was then disabled. Independent training of SKAN on randomly selected sections of the dataset consistently resulted in similar spatio-temporal features being learnt. The panels in **Figure 8** show the resulting feature set from two independent trials at different network sizes to demonstrate this. As the comparison of the trained feature sets shows the same consistent features were learnt at each network size, with the features coding for the leading edge of the airplane nose cones and wings dominating the feature sets. In addition, variants of a solitary noise spike produced often by the ATIS camera are represented as noise features appearing in top left of **Figures 8B–D**. This consistency was also observed over training epochs of the individual trials. As the number of neurons is increased some of the neurons no longer code for the same features, as can be best seen in the bottom right neurons of **Figure 8D**. Note also the increasing number of variants of the "noise feature" as the network size is increased. These variants of the "noise feature" encode weak traces of features which are too weak to show in the full color scale.

Of the many network sizes shown in **Figure 8** the 25 neuron network was chosen for the investigation of the other parameters in the system. In section Feature Extractor Size and Number, we return to investigate the effect of network and feature sizes in greater detail. Following feature extraction, and with learning disabled, the neurons compete to recognize incoming spatio-temporal event patterns generated from the same 13 × 13-pixel region of the surface following each new event with the spike output of the winning neuron representing a feature event. These feature events were then stored onto 25 separate *feature time surface or feature index surfaces*, which were generated identically to the event surfaces described in section Time-Surface vs. Index Surfaces using the same decay method and decay factor.

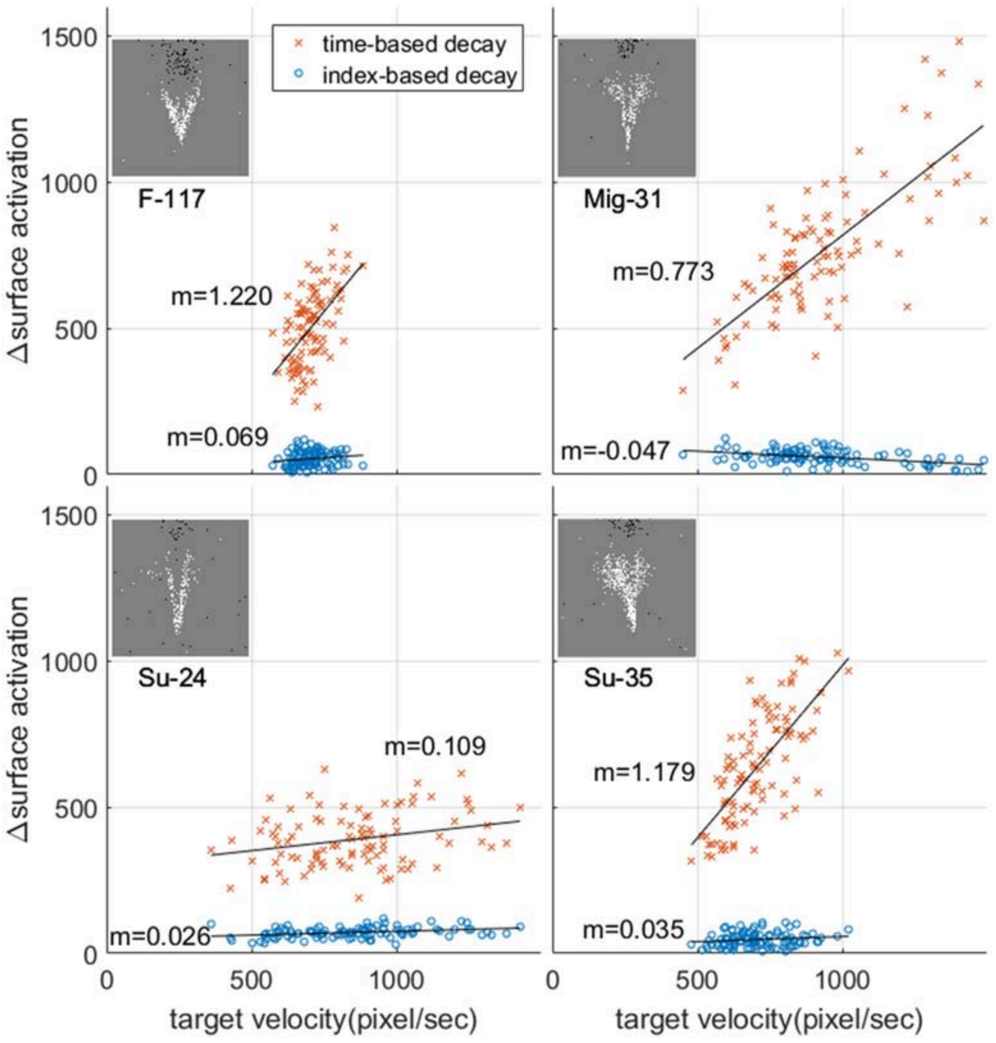


FIGURE 7 | Relationship between change in surface activation and target velocity and the resultant mean rate of change in surface activation. Each point represents a single recording in the dataset. The mean value of target velocity and change in surface activation was calculated over the time window indicated in **Figure 6 (A)**. For each panel m indicates the slope of the line of best fit.

Spatial Pooling of Feature Surfaces

In order to reduce the required processing and speed up simulation, the subsystems following the feature surfaces were operated in a frame-based manner such that at periodic intervals the estimated target region from each feature surface was sampled to generate feature frames. The interval used for sampling was the same as the time surface decay constant $\tau_e = 3\,\text{ms}$. The surface sampling was time-based for both the time and index surfaces so as not to bias the comparison. To reduce the input size to the classifier, spatial pooling of the feature surfaces was performed. To perform this spatial pooling, the estimated object boundary region was summed along the rows and columns, generating two one dimensional feature vectors, one for the rows and one for the columns. The length of these vectors would vary at each feature frame depending on the size of the estimated target region. Thus, in a network with N neurons for each feature a target region of size R rows and C columns would generate two one-dimensional vectors (of length R and C, respectively) resulting from the summation of the image region across rows and columns for each of N surfaces. In order to provide the classifier with a uniform input layer size, the varied length feature vectors R and C need to be resampled to a uniform length. This was done using linear interpolation and the uniform vector length chosen was 72, which, when multiplied by the number of pooling dimensions (2), and the number of features (25), produced a 3,600-input layer for the classifier. The resultant end-to-end system is shown in **Figure 9**.

Parameter Selection

In order to fairly evaluate the relative performance in terms of recognition accuracy resulting from different decay kernels, surfaces decay methods, feature extractor numbers, and their

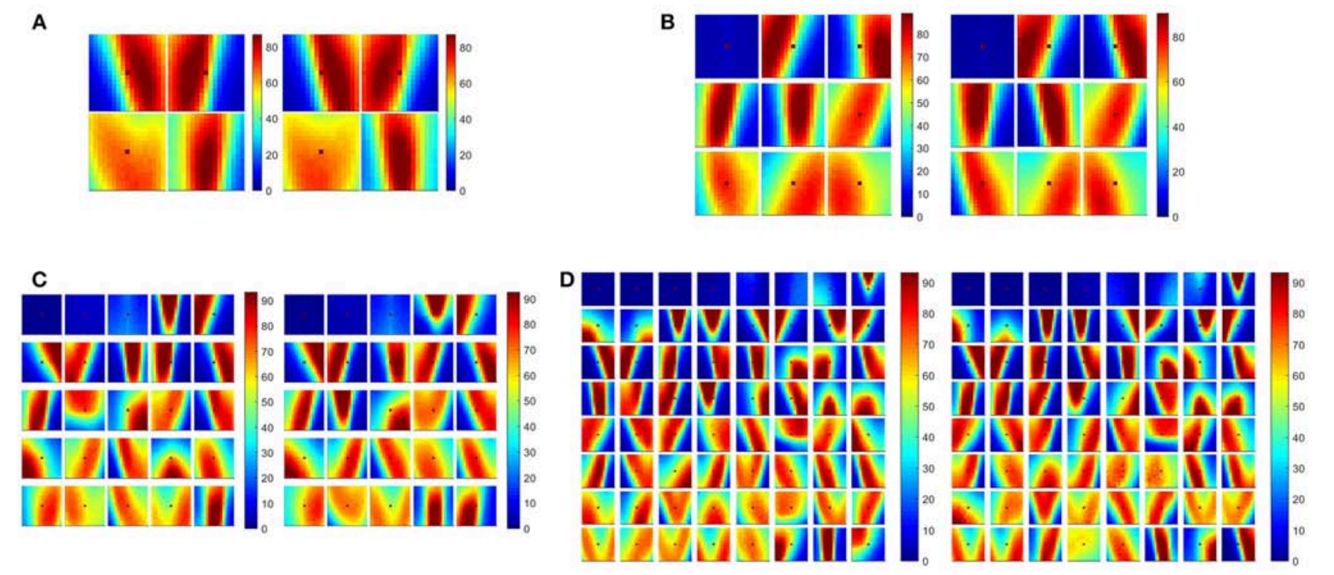


FIGURE 8 | Consistency of feature generation at multiple network scales. **(A–D)** show 4, 9, 25, and 64 spatio-temporal features, respectively, extracted from the ATIS airplane drop dataset. Each panel show results from two independent trials. To allow for visual comparison of the two feature sets, the features from the first trial have been ordered based on the sum of the squares of the weight of each pixel in each feature. The features of the second trial were then sorted based on cosine distance to the first feature set. Only the feature-set obtained from two instances of the time-based, exponentially decaying surface is shown above for brevity. The features resulting from the other kernels resulted in qualitatively similar features dominated by wing edge, nose cone tail features as well as features coding for noise.

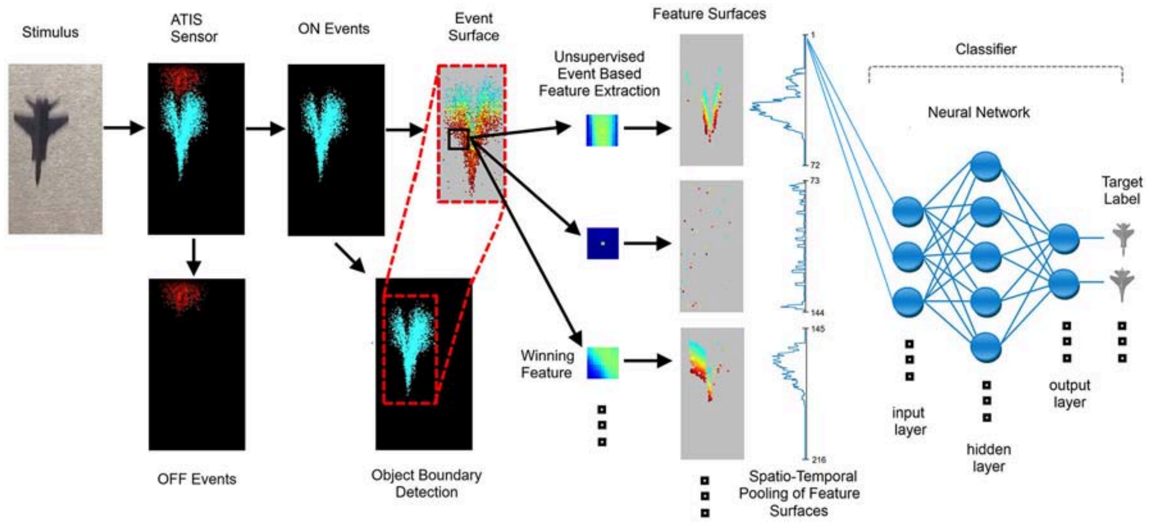


FIGURE 9 | Block diagram of the full event-based detection feature extraction and recognition system. The target is sensed by the sensor and the generated ON events are processed using a time or index surface. Each event triggers a comparison of a local patch around the event with a set of features or neurons. The winning neuron outputs an event which in turn is placed on a feature surface. The feature surfaces are summed across the rows and columns and presented to the back end classifier. The classifier is here depicted as a network with a hidden layer but we also use a linear classifier. Note that in the feature surface pooling stage only the vector summing the feature surfaces across columns is shown, with the second vector showing the summation across rows omitted for clarity.

receptive field sizes, a large number of free system parameters must first be selected. These parameters, listed in **Table 1**, are used to implement event and feature surface generation, surface sampling, object detection, feature extraction, spatial pooling, regularization, and classification. In order to ensure that the selected parameters do not advantage the index-surfaces or the feature extraction methods that are the focus of this work, all subsystem parameters would need to be evaluated in terms of their combined effects on the performance of each method under testing. However, this represents a prohibitively large search space to explore in a brute force fashion. Instead, the approach taken in this work to remove possible parameter

TABLE 1 | Free parameters used in the system (unless otherwise stated).

Subsystem	Parameter	Value
Surface generation	Time constant	$\tau_e = 3$ ms
Surface generation	Index constant	$N_e = 554$ events
Detector	Smoothing window size	8 pixels
Detector	Smoothing window type	Moving average
Detector	Normalized threshold	0.1
SKAN	Number of features	25
SKAN	Number of input channels	$13 \times 13 = 169$
SKAN	Other parameters	Same as Afshar et al. (2014)
Classifier	Input size using raw event surface (E)	$72 \times 2 = 144$
Classifier	Input size feature event surfaces (F)	$72 \times 25 \times 2 =$ 3,600
Classifier	ELM hidden layer size	30,000 Neurons
Classifier	Surface sampling interval	3 ms

selection bias in favor of the proposed methods was to optimize all parameters to achieve the highest recognition accuracy on what may be considered the null hypothesis: that simple time-based binning kernels used on raw input events outperform other kernels, decay methods, and feature extractors. To this end, the parameters in **Table 1** and all algorithm design choices where selected via a manual heuristic search for optimal recognition performance using the time-based binning surface BTS_i whose spatially pooled output was fed directly to the classifier without the use of feature extractors. The classifiers were then selected for optimal performance on the output data generated by the selected parameters. Once optimized in this way for the "null hypothesis," these same parameters and network structures were used for all other tests, ensuring that recognition results were biased in favor of the simple time-based binning approach and not those proposed in this work.

Classification

Choosing Classifiers

The choice of a back-end classifier used to map feature outputs to classes can play a critical role in the performance of a convolutional feature extraction layer or network. Well-regularized high capacity classifiers with internal non-linearities can provide significant improvement in performance over and above the underlying feature extractors used. In many proposed event-based recognition systems, only a single type of classifier is tested and often only a single instance of such a classifier (the best performing configuration) is reported. While this approach encourages greater attention to the presented work, it can also overstate the performance of the overall system, due to fine tuning. What's more, the use of well-optimized powerful classifiers without concurrently testing simple linear classifiers obscures the role of the event-based feature extractors in the system performance. Here, we propose a dual classifier testing protocol, which ideally should be applied before and after each stage of processing, to provide insights into the effectiveness of the elements under test. For the baseline test, a simple linear classifier is used to measure how linearly separable the underlying data is before and after processing. In addition to this baseline classifier we utilize a large capacity ELM, which, by virtue of the large number of random hidden layer neurons, is likely to project the non-linearities of the dataset into a linearly separable higher dimensional feature space. In addition, the lack of feature learning in the ELM allows a reasonable unbiased estimate of the residual non-linearity in data. This framework of testing provides significant insights, as detailed in the results section, which would not be revealed if only the results from the best performing classifier were reported.

To evaluate the performance of the system, two measures of recognition accuracy were considered: per-frame accuracy and per-drop accuracy. For the per-frame measure, the feature vectors described Section Event-Based Feature Extraction were presented to the classifier at periodic time intervals τ_e. At each frame, the class with the largest output was selected as the winner for that frame. For the per-drop accuracy measure, the class with the highest number of per-frame during the entire recording was selected.

A linear classifier and an Extreme Learning Machine (ELM) classifier (Cohen et al., 2017) with a hidden layer size of 30,000 neurons were trained using the time-based binning method to achieve the highest per-frame recognition accuracy. **Figure 10** details the results from this parameter search and the selected classifiers.

RESULTS

Results on the Full Dataset

The per-frame recognition results on the full dataset are shown in **Figure 10**. For each of the panels, the same performance pattern is observed: when operating on raw event surfaces as inputs, the large capacity ELM (ELM-E) significantly outperforms the linear classifier (L-E). This demonstrates the non-linearity of the classification boundaries in this case. In comparison, when feature surfaces are used as inputs, the improvement margin gained by the ELM (ELM-F) is small relative to the linear classifier (L-F) suggesting that the output of the 25 feature extractors is significantly more linearly separable, with less room for improvement through further non-linear expansion. Also noteworthy is that the linear classifier operating on feature surfaces (L-F) outperforms the ELM operating on the event surfaces (ELM-E) for all surfaces generation methods. This shows that the application of a small number of trained local feature extractors is more effective than using a much larger globally connected network of neurons with random input weights. The ratio of errors between the ELM and the linear classifier indicated at the bottom of each panel quantifies this reduction in error for each case.

Comparing the results across the panels for the linear classifier operating on events (L-E), the exponentially decaying surfaces outperform linear surfaces by a margin of 1.75% for the index surfaces and 0.24% for the time-surfaces. In turn the linear surfaces outperform the binning method by 3.06 and 1.36% for the index surfaces and time surfaces, respectively. For the case of the linear classifier operating on feature surfaces (L-F), the

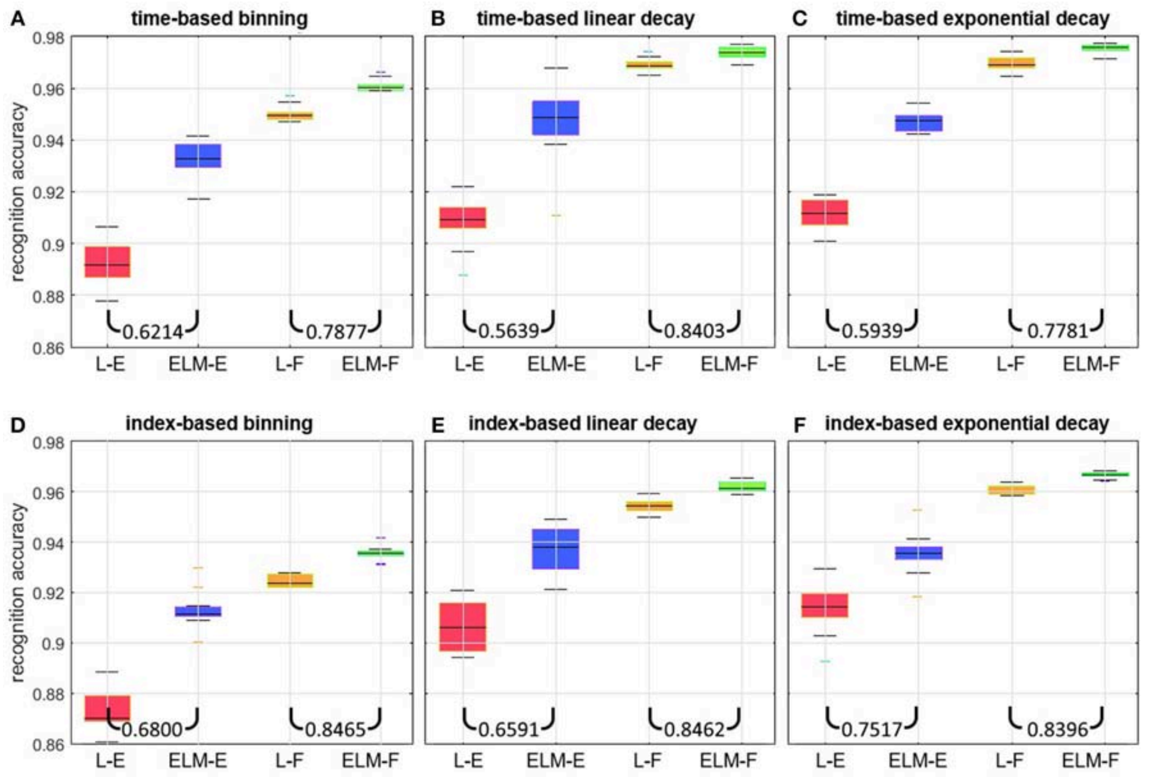


FIGURE 10 | Per-frame recognition accuracy on the full dataset over $n = 20$ independent trials. Each panel shows results from four network arrangements. In (L-E), and (ELM-E) the linear classifier and the 30 K hidden layer ELM chosen in section Choosing classifiers operate on inputs from raw event surfaces. In (L-F), and (ELM-F) the same classifiers use 25 feature surfaces as inputs. Each panel shows results for a different surface generation method: The top three panels show time-based methods using **(A)** binning, **(B)** linear decaying, and **(C)** exponentially decaying surfaces. The bottom three panels show corresponding index-based binning **(D)**, linear decaying **(E)**, and exponentially decaying surfaces **(F)**. The two ratios at the bottom of each panel indicate the median error ratio of the ELM over the linear classifier.

exponentially decaying surfaces outperform linear surfaces by a margin of 0.57% for the index surfaces and 0.22% for the time-surfaces, and in turn the linear surfaces outperform the binning method by 3.07 and 1.91% for the index surfaces and time surfaces, respectively. Also, consistently, the improvement of exponential kernels over linear kernels is not as significant as their margin with the binning method.

It is worth noting that, when the ELM is chosen as the back-end classifier, the margin in performance improvement obtained from feature extraction is reduced. This is to be expected, since the randomly situated hidden layer neurons of the ELM have a greater chance of improving the linear separability of segments of the dataset, if such segments are not already linearly separable due to processing in the preceding layer. This effect of obscuring the performance of other subsystems is not limited to the ELM. A similar effect would be expected with any other classifier performing non-linear expansion. This underlines the need to include results from a simple linear classifier when comparing alternative systems. Also worth noting is that for the preceding results (features outperforming raw events, and exponential and linear kernels outperforming binning) all system parameters were optimized for the time-based binning method. These results therefore confirm the suitability of exponential kernels for time and index-surface generation. This conclusion is also supported by results in Akolkar et al. (2015), where the information from the visual scene is found to rapidly rise within a small initial temporal window, but thereafter fall gradually with increasing window size, as is best described by an exponentially decaying kernel. By weighing events in an approximately compensatory manner to their information content as described in Akolkar et al. (2015), the exponentially decaying kernel results in the highest information content for the classifier. Another observation from **Figure 10** is that all time-based decay methods outperform the index-based decay methods by ∼1% on the full dataset with the largest performance disparity observed between the index-based binning method BIS_i and the time-based binning method BTS_i. This would be expected, since the later method was used during all parameter optimizations and would be most advantaged by the selected parameters. Based on the results shown in **Figure 10** we narrow further investigations by selecting linear classifiers L-E and L-F and focus on exponentially decaying surfaces EIS_i and ETS_i.

Frame Balanced Dataset

In order to generate a balanced dataset, an equal number of frames from each recording was selected. In this way, the total number of presentations to the classifier for each class was equalized. As **Figure 11** shows 1, 2, 4, 8, 16, and 32 frames were sampled from each of the airplane recordings and presented to the linear classifier operating on events surfaces L-E and feature surfaces L-F for each of the EIS_i and ETS_i surfaces.

As **Figure 11** shows, both the per-frame and per-drop accuracy increase as a function of the number of frames used during training. Additionally a sharper increase and higher final accuracy is observed for the per-drop accuracy measure, as would be expected, since the per-drop measure is analogous to a max pooling operation which benefits from increased pool size. The relative performance margin of the network using feature surfaces over raw event surfaces is reduced in the per-drop measure, as more information is accumulated over a recording, reducing error, and approaching the 100% accuracy upper bound. The highest number of random frames used per recording was 32, as this was approximately equal to the total number of frames in the shortest recording (see **Figure 2B**). **Table 2** details the accuracy results for this balanced dataset while **Figure 12** shows misclassified recordings for one instance of the highest performing network using index-based decaying feature surfaces and a linear classifier, illustrating that some drops are almost impossible to classify correctly.

Interestingly, in contrast to the full unbalanced dataset results detailed in section Results on the Full Dataset, the per frame balanced results in **Figure 11** and **Table 2** show little significant difference in accuracy between the index-based and time-based surfaces for either the per-frame or per-drop measures, suggesting that the observed slight advantages in accuracy on the full dataset may be due to the use of time-based surfaces during parameter selection of section Parameter Selection and linked to imbalances in the number of frames per recording present in the full dataset for the two different methods.

Velocity Segregated Dataset

As outlined in section Target Velocity vs. Surface Activation, the apparent velocity invariance property of index surfaces motivates a test using a modified dataset which is split in terms of target velocity. Thus, in order to compare index-based and time-based surfaces in terms of target velocity invariance, the recordings were divided into 200 "slow" and 200 "fast" recordings based on the estimated vertical airplane velocity at the midpoint (in time) of each recording. Since the airplanes speed up during

TABLE 2 | Per-frame and Per-drop accuracy results on the frame balanced dataset for four selected systems: Linear classifier operating on events surfaces (L-E) and feature surfaces (L-F) for each of the $\mathbf{EIS_i}$ and $\mathbf{ETS_i}$ surfaces.

	Per-frame (%)	**Per-drop (%)**
Time-based Event surface	90.60 +/−1.02	96.64 +/−1.47
Index-based Event surface	91.03 +/−0.89	96.90 +/−1.34
Time-based Feature surfaces	95.64 +/−0.79	98.52 +/−0.75
Index-based feature surfaces	96.15 +/−0.84	98.75 +/−0.78

Number of trials used is 20.

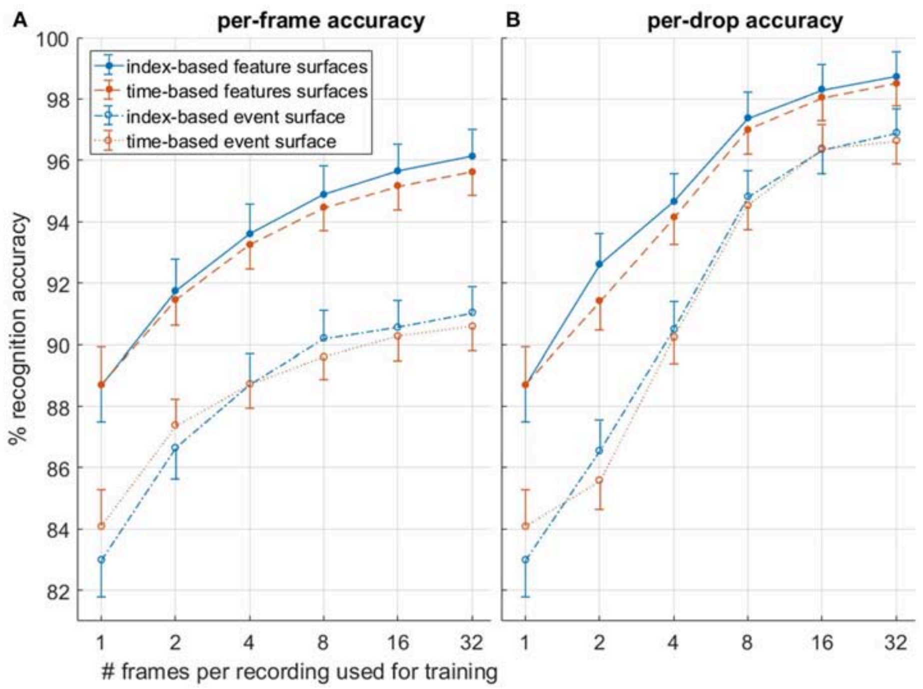


FIGURE 11 | Comparison of **(B)** per-drop and **(A)** per-frame recognition accuracy as a function of the number of randomly selected frames used during training from each recording. The index-based EIS_i surface and time-based ETS_i surfaces are compared. Results shown are over $N = 20$ trials. A linear classifier was used in all test.

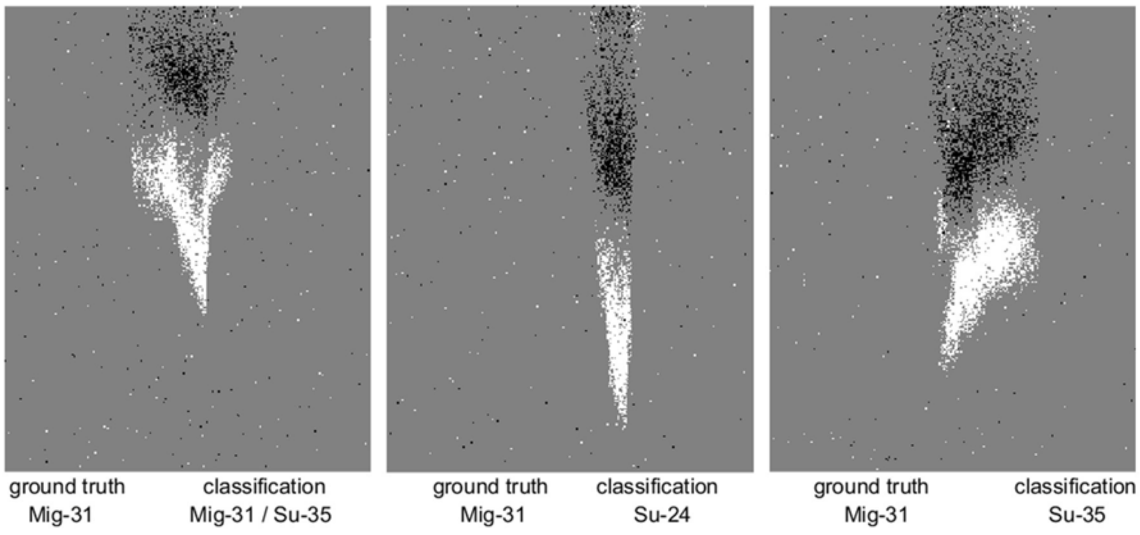


FIGURE 12 | The three drops misclassified by an instance of a linear classifier using 25 exponentially decaying index-based feature surfaces. Captured frames show airplanes at mid-point (in time) of recording.

the fall, the system was trained on the n-first (slowest) frames of the slow recordings and tested on the n-last (fastest) frames of the fast recordings. In this way, by varying the number of frames n, datasets with different degrees of velocity segregation could be tested. The resulting recognition accuracies in **Figure 13** demonstrate that with increasing n, and thus decreasing velocity segregation in the data, the recognition accuracy of all systems rise. **Figure 13** further shows that although training on a speed segregated dataset significantly reduces accuracies for all systems in comparison to training using a randomly sampled dataset (such as shown in **Figure 11**), the decline is significantly larger for time-based decaying surfaces. This difference demonstrates the relative robustness of index-based decay surfaces to variance in velocity and their utility in applications where the full range of potential target velocities to be encountered during testing is not available in the training data.

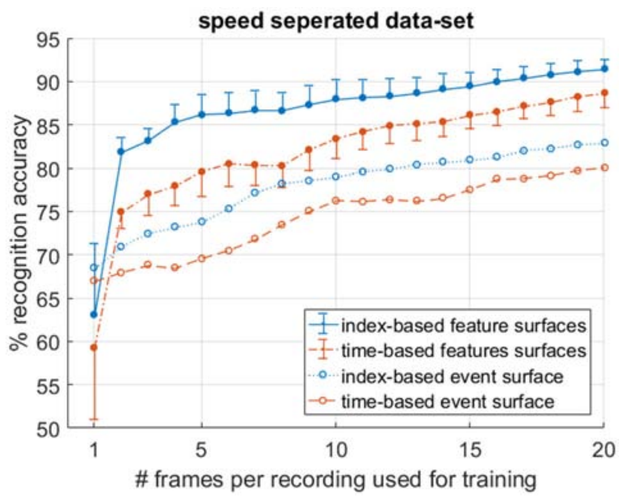


FIGURE 13 | Mean and standard deviation per-frame accuracy on a speed segregated dataset over 10 trials clearly demonstrates superior performance of index-based surfaces in the presence of velocity varying data.

Therefore, given the results in the previous section, it can be concluded that, at the local scale, with a single target in the field of view, systems using index-based decay surfaces tend to match equivalent systems using time-based decay surfaces, when presented with an adequately wide range of velocities in the training data, since their advantage of velocity invariance is effectively neutralized. But when the available range of velocity distributions for training is incomplete, index-based decay surfaces tend to produce more robust performance. Given this finding, and in order to limit the scope in the next section, we narrow our focus exclusively on index-based surfaces and investigate the effect of different feature extraction networks and their effect on recognition accuracy. This is also supported by findings in Ghosh et al. (2014), where a small superiority was found when using fixed event windows over time windows. However, those tests were performed using a randomly sampled training set, likely containing data with velocity distributions that were similar to the test set. As such their results are similar to the full dataset results examined in section Frame Balanced Dataset of this work, which only showed a slight improvement due to the velocity variance available in the training dataset. In this work, by additionally testing the algorithms using a range of velocity segregated datasets, the robustness of the index surface method is more completely investigated.

The Decay Constants

An important element of any event-based surface is the value of its decay constant. In this work the value of decay constants, $\tau_e = 3$ ms and $N_e = 554$ events were effectively chosen arbitrarily.

This raises an important question about the optimality of the chosen decay constants and the robustness of the generated features and recognition accuracy to different values of these decay constants. A closely related question, which applies only to index surfaces, is whether targets which generate more or fewer events, e.g., due to different object size or contrast, could still be learnt and recognized with the decay constants chosen. To investigate these questions a wide range of decay constants across six orders of magnitudes were tested on a frame balanced randomized training and testing dataset. The resulting recognition accuracies and selected feature sets are shown in **Figure 14**. The results show a similar pattern for time and index surfaces with little significant difference in accuracy. At the extreme decay rate of 10 events and 54 µs the systems perform little better than chance, since virtually all event information is decayed away before it can be extracted. This leaves all the features coding for variants of the noise feature. As the decay constant increases to by two orders of magnitude, coherent features begin to emerge coinciding with a rapid increase in recognition accuracy. At this event rate there are still multiple features coding for a single noise spike. Index decay constants of between three and four orders of magnitude of events correspond with a plateau in recognition performance. This region coincides with the range where the noise feature is only represented by one or two neurons with all remaining neurons coding for complex features. After four orders of magnitude increase in the decay constant, the accuracy begins to decline slightly. In this region the noise features begin to be represented once more but this time with a highly activated background which is a direct result of the much slower decay rate.

As **Figure 14** illustrates, when sweeping the decay constant, the number of variants of the noise feature in the network roughly correlates to the feature extraction performance of the network. The feature set with the fewest representations of the noise feature (ideally only one) performs the best. This is expected since the noise feature is unlikely to be correlated to any particular class of object and the frequency of its representation in a feature-set reduces the efficiency of that feature-set, leaving fewer neurons

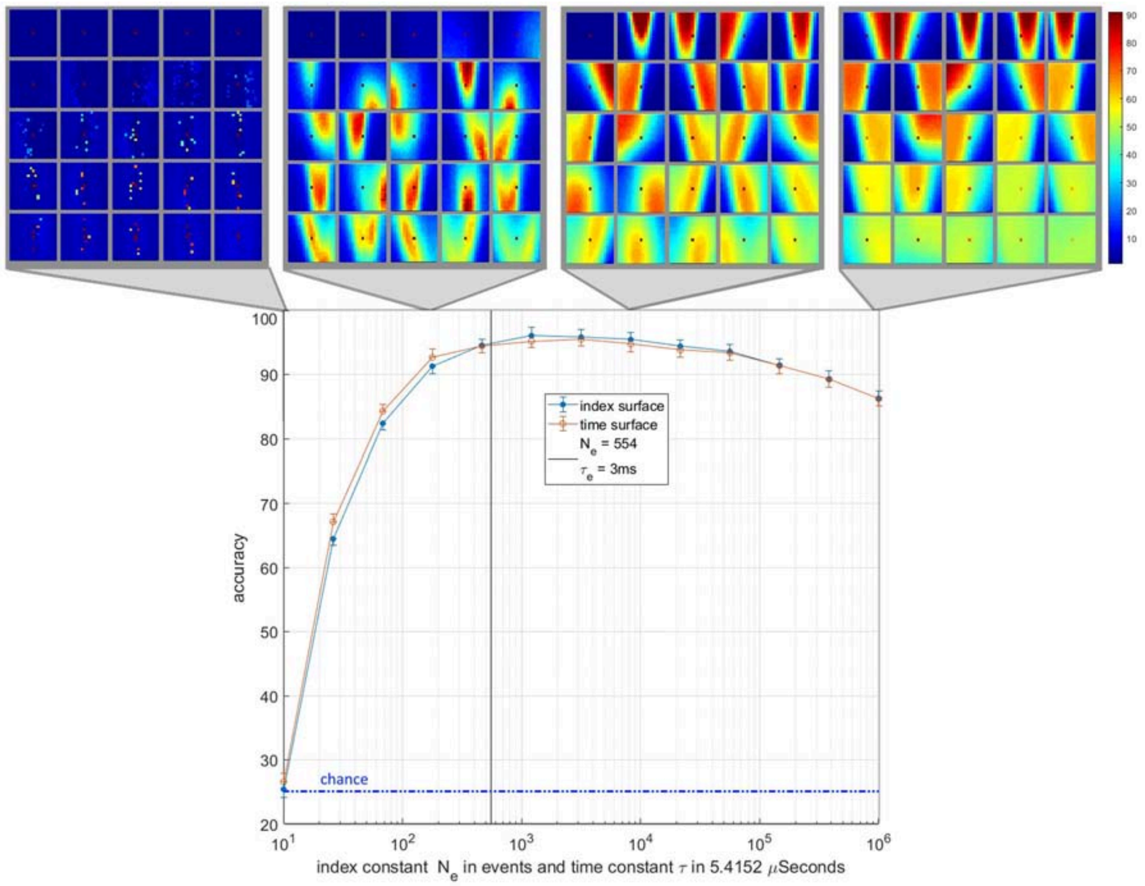


FIGURE 14 | Classification accuracy and typical feature sets as a function of the decay constants for time and index surfaces. The lower panel shows accuracy plotted against the index decay constant $N_e$on a logarithmic scale. The time surface results are plotted on the same logarithmic scale where a 1event to 5.4152 µs conversion rate is used to align the results. This conversion rate is based on the average event rate over the entire dataset. The vertical solid line at $N_e = 554$ and $\tau_e = 3$ ms ($\tau_e = 554 \times 5.4152$ µs) indicates the value of the index and time decay constants used in rest of the work. The horizontal dotted line indicates chance accuracy. All tests were performed over $N = 20$ independent feature extraction trials. The feature sets above the panel show instances of the feature sets for four points on the decay constant axis. The feature set shown are from index-based surfaces.

to represent classification relevant feature information. **Figure 14** also shows a wide central region of stable performance that is robust to the choice of τ_e and N_e. The results also show that over estimating the optimal value of the decay constant is less harmful than under estimating with significantly less reduction in accuracy.

Feature Extractor Size and Number

In order to characterize the effectiveness of the feature extraction subsystem in an unbiased manner, a range of feature sizes and a number of feature extractors were investigated and assessed in terms of the resultant recognition accuracy. In addition, for each point on the feature size-feature number space, the results of the learning algorithm described in section Event-Based Feature Extraction was compared to those of equivalent sized networks using random feature sets. The mean accuracy results in **Figure 15** (top panels) demonstrate that learnt features outperform random features at every scale while exhibiting slightly lower variance in accuracy (bottom panels).

In addition, while the results from the random features suggest a slight trend toward increased accuracy as a function of both feature numbers and feature size, the learnt feature results clearly show that the larger feature sizes (17 × 17 and 13 × 13) generate higher accuracy with increasing number of features, while the smallest feature sizes (3 × 3 and 5 × 5) exhibits a weak downward trend with the number of features. When the feature size is small, only a few distinct combinations exist.

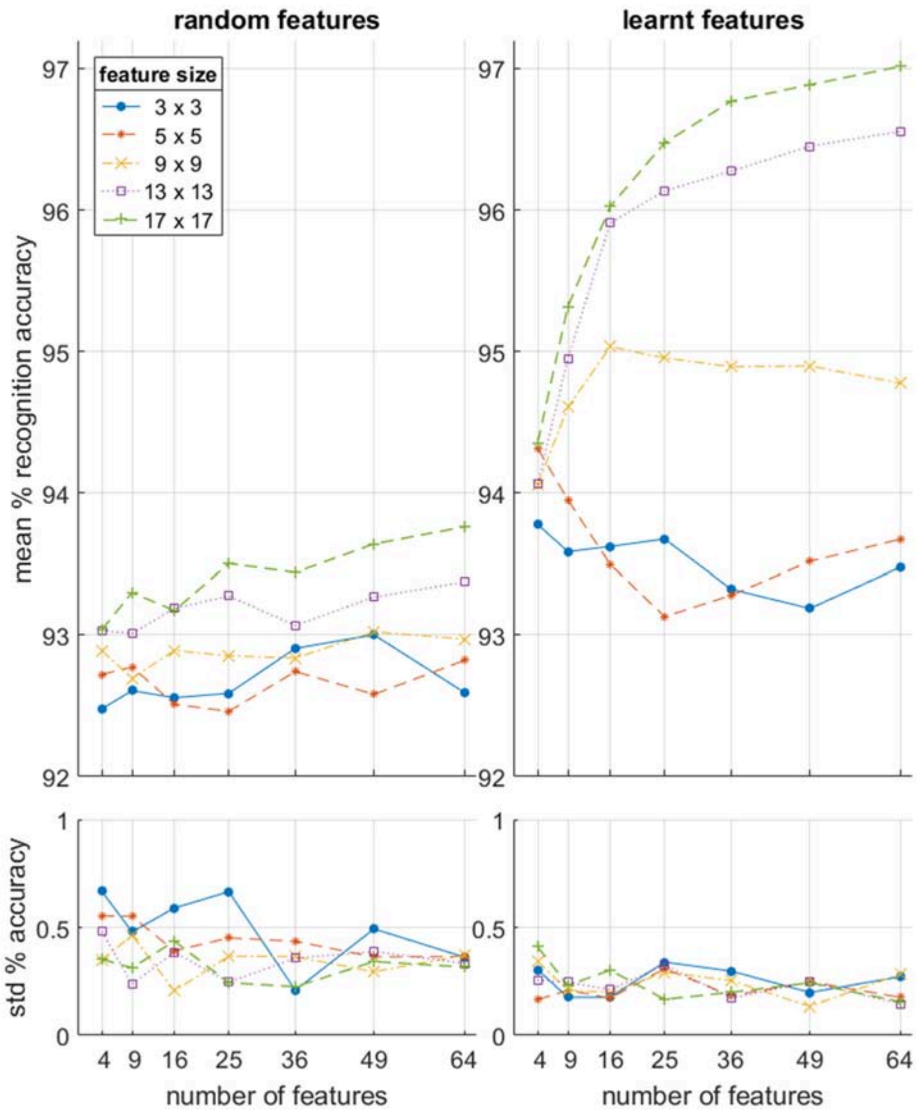


FIGURE 15 | Per-frame accuracy on the full dataset as a function of feature size and number of features used in the feature extraction layer for both learnt and random features. $N = 10$ independent feature sets with 10 cross validating classifications per feature set. Note that the baseline linear accuracy using the raw event surface with no feature extraction layer was 91.38 +/− 0.81% as shown in **Table 2**.

Therefore, when and a large number of them are trained, several features will be very similar, resulting in near identical input generating very different input to the classifier. This reduction in accuracy resulting from the addition of more redundant features is due to the OR operation which must be performed by the back-end classifier. This insight demonstrates that convolutional features layers can, if poorly configured, "over-fit" the data by representing overly specific variants of the same pattern. This effect only becomes apparent with the combined use of a large number of feature, small feature sizes, and relatively small datasets. But this might be an issue in future applications of event-based convolutional networks, where resource efficiency of a hardware implementation may allow a very large number of features in a layer to be trained (especially in the first layer) while the level of independent features in the recorded data may be limited.

We can also note that for both the random and learnt feature sets, the feature size has little effect on accuracy when the number of features becomes very small. This is because there is very little additional discriminatory information that can be captured by the larger sized features when a wide range of unrelated, heterogeneous spatio-temporal patterns become effectively averaged together to generate the (too) few features used in the network. Thus, local spatial complexity of observed data determines optimal feature size and feature number relationships, which, if not considered during hardware implementation, can result in inappropriately scaled network architectures and effectively wasting hardware resources.

DISCUSSION

While binning methods examined in this work were shown to perform less well than linearly decaying surfaces and exponentially decaying surfaces, the significantly simpler implementation of the binning method allows for much more efficient implementations of event surfaces in neuromorphic hardware. In a similar fashion, the selection of feature sizes and number of features implemented at any layer of a multi-layer event-based network generates trade-offs between hardware resource and performance. In this context, the network and feature size investigations presented here provide guidelines for such network designs.

The four class dataset presented allows reasonably accurate classification using a single layer of feature extraction in combination with a linear classifier; the task can be made increasing difficult by increasing the number of classes in the dataset. In such a case the output of the feature extraction layer would retain significantly greater residual non-linearity. This would increase the performance of gap between the linear classifier and the large ELM. Conversely adding additional feature extraction layers will work in the opposite direction, producing output that is more and more linearly separable and thus reducing the performance gap between the linear classifier and ELM.

The presented recordings in the dataset were varied to cover a wide range of target speeds. As a result any random splitting of training and testing data provided an overlapping range of target speeds in both set. This overlap removed any advantage of index-based decaying surfaces which provide robustness to target velocity. However, in many applications, such as the SSA applications of Cohen et al. (2017), the range of velocities in the training set is limited so that features trained on this limited set of target velocities must generalized to a wide range of as yet unobserved velocity profiles. In this work, such a condition was simulated by iteratively segregating the data based on speed to highlight the utility of the index-based decay method.

One weakness of the index-based decaying method is that it can only be used locally (or globally but on a single target). If events from other non-target object cause a decay in the surface activation of the target, vital information may be lost. Such information loss is not present if target segregation has already occurred via an upstream system, or, more generally, if the surface decay mechanism is viewed as a local mechanism acting on a sub-region of a larger global surface. As such, the presented dataset and the resulting performance of the index-based systems can best be viewed as focusing on a locally operating subsystem within a larger processing system. When viewed as a rigorous analysis of such a central building block of a larger event-based network the value of the investigation presented here becomes more apparent. On the other hand, if a system needs to operate with a single decay method, then the standard time-based decay mechanism would be more optimal, as it can process the entire surface in a global manner.

CONCLUSION

In this work, we investigated in detail an event-based feature extraction layer. In order to rigorously investigate the effects of different kernels, decaying methods, classifiers, and feature sizes and numbers, we limited the exploration to a single layer network. Yet the design of deeper networks can be informed by these single layer results. Using a dataset featuring a range of target shapes, scales, orientations, and velocities, it was observed that exponentially decaying kernels outperform other kernels, and that index-based decaying surfaces perform equally as well as time-based decaying surfaces, when robustness to target speed is not required, and outperform them when it is required. We also showed a clear superiority of learnt features over random features and showed that the largest networks of neurons with the largest receptive fields using the most complex kernels outperform all other configurations.

AUTHOR CONTRIBUTIONS

SA, GC, and TH designed dataset. SA and GC generated the dataset. SA and GC performed pre-processing. SA, GC, JT, and AvS designed the algorithms. SA implemented the algorithms. SA analyzed the data and results. SA wrote the manuscript. All authors assisted in editing.

REFERENCES

Afshar, G., Hamilton, S., Tapson, T. J., van Schaik, J., and Cohen, A. (2018). *ATIS Plane Dataset*. Available online at: https://www.westernsydney.edu.au/bens/home/reproducible_research/atis_planes

Afshar, S., George, L., Tapson, J., van Schaik, A., and Hamilton, T., J. (2014). Racing to learn : statistical inference and learning in a single spiking neuron with adaptive kernels. *Front. Neurosci.* 8:377. doi: 10.3389/fnins.2014.00377

Afshar, S., George, L., Thakur, C. S., Tapson, J., van Schaik, A., de Chazal, P., et al. (2015). Turn down that noise: synaptic encoding of afferent SNR in a single spiking neuron. *IEEE Trans. Biomed. Circuits Syst.* 9, 188–196. doi: 10.1109/TBCAS.2015.2416391

Akolkar, H., Meyer, C., Clady, Z., Marre, O., Bartolozzi, C., and Panzeri, S. (2015). What can neuromorphic event-driven precise timing add to spike-based pattern recognition? *Neural Comput.* 27, 561–593. doi: 10.1162/NECO_a_00703

Barranco, F., Fermuller, C., Aloimonos, Y., and Delbruck, T. (2016). A dataset for visual navigation with neuromorphic methods. *Front. Neurosci.* 10:49. doi: 10.3389/fnins.2016.00049

Benosman, R., Ieng, S. H., Clercq, C., Bartolozzi, C., and Srinivasan, M. (2012). Asynchronous frameless event-based optical flow. *Neural Netw.* 27, 32–7. doi: 10.1016/j.neunet.2011.11.001

Clady, X., Ieng, S. H., and Benosman, R. (2015). Asynchronous event-based corner detection and matching. *Neural Netw.* 66, 91–106. doi: 10.1016/j.neunet.2015.02.013

Cohen, G., Afshar, S., van Schaik, A., Wabnitz, A., Bessell, T., Rutten, M., et al. (2017). "Event-based Sensing for Space Situational Awareness," in *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)* (Maui, HI), 1–13.

Ghosh, R., Mishra, A., Orchard, G., and Thakor, N. V. (2014). "Real-time object recognition and orientation estimation using an event-based camera and CNN," in *IEEE 2014 Biomedical Circuits and Systems Conference, BioCAS 2014 - Proceedings* (Lausanne), 544–547.

Giulioni, M., Corradi, F., Dante, V., and del Giudice, P. (2015). Real time unsupervised learning of visual stimuli in neuromorphic VLSI systems. *Sci. Rep.* 5:14730. doi: 10.1038/srep14730

Glover, A., and Bartolozzi, C. (2016). "Event-driven ball detection and gaze fixation in cluttter," in *IEEE International Conference on Intelligent Robots and Systems* (Daejeon), 2203–2208.

Glover, A., and Bartolozzi, C. (2017). "Robust visual tracking with a freely-moving event camera," in *IEEE International Conference on Intelligent Robots and Systems* (Vancouver, BC), 3769–3776.

Hu, Y., Liu, H., Pfeiffer, M., and Delbruck, T. (2016). DVS benchmark datasets for object tracking, action recognition, and object recognition. *Front. Neurosci.* 10:405. doi: 10.3389/fnins.2016.00405

Klein, P., Conradt, J., and Liu, S. C. (2015). "Scene stitching with event-driven sensors on a robot head platform," in *2015 IEEE International Symposium on Circuits and Systems* (Lisbon: ISCAS), 2421–2424.

Lagorce, X., Ieng, S. H., Clady, X., Pfeiffer, M., Benosman, R., et al. (2015a). Spatiotemporal features for asynchronous event-based data. *Front. Neurosci.* 9:46. doi: 10.3389/fnins.2015.00046

Lagorce, X., Meyer, C., Ieng, S. H., Filliat, D., and Benosman, R. (2015b). Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1710–1720. doi: 10.1109/TNNLS.2014.2352401

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Analy. Mach. Intell.* 39.7, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lee, J. H., Delbruck, T., Pfeiffer, M., Park, P. K. J., Shin, C.-W., Ryu, H., et al. (2014). Real-time gesture interface based on event-driven processing from stereo silicon retinas. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 2250–2263. doi: 10.1109/TNNLS.2014.2308551

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128× 128 120 dB 15 µs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Litzenberger, S., and Sabo, A. (2012). Can silicon retina sensors be used for optical motion analysis in sports? *Proc. Eng.* 34, 748–753. doi: 10.1016/j.proeng.2012.04.128

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213–215. doi: 10.1126/science.275.5297.213

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3, 0247–0257. doi: 10.1371/journal.pcbi.0030031

Orchard, G., Jayawant, A., Cohen, G., K., and Thakor, N. (2015a). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015b). "HFirst: a temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 2028–2040. doi: 10.1109/TPAMI.2015.2392947

Peng, X., Zhao, B., Yan, R., Tang, H., and Yi, Z. (2017). Bag of events: an efficient probability-based feature extraction method for AER image sensors. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 791–803. doi: 10.1109/TNNLS.2016.2536741

Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with ldossless pixel-level video compression and time-domain CDS. *IEEE J. Solid State Circuits* 46, 259–275. doi: 10.1109/JSSC.2010.2085952

Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Front. Neurosci.* 9:481. doi: 10.3389/fnins.2015.00481

Sofatzis, R. J., Afshar, S., and Hamilton, T. J. (2014). "Rotationally invariant vision recognition with neuromorphic transformation and learning networks," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)* (Melbourne, VIC), 469–472.

Zhao, B., Ding, R., Chen, S., Linares-Barranco, B., and Tang, H. (2015). Feedforward categorization on AER motion events using cortex-like features in a spiking neural network. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1963–1978. doi: 10.1109/TNNLS.2014.2362542

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 22 February 2019
doi: 10.3389/fnins.2019.00004

REMODEL: Rethinking Deep CNN Models to Detect and Count on a NeuroSynaptic System

Rohit Shukla[1]*, *Mikko Lipasti*[1], *Brian Van Essen*[2], *Adam Moody*[2] *and Naoya Maruyama*[2]

[1] *Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI, United States,*
[2] *Lawrence Livermore National Laboratory, Livermore, CA, United States*

In this work, we perform analysis of detection and counting of cars using a low-power IBM TrueNorth Neurosynaptic System. For our evaluation we looked at a publicly-available dataset that has overhead imagery of cars with context present in the image. The trained neural network for image analysis was deployed on the NS16e system using IBM's EEDN training framework. Through multiple experiments we identify the architectural bottlenecks present in TrueNorth system that does not let us deploy large neural network structures. Following these experiments we propose changes to CNN model to circumvent these architectural bottlenecks. The results of these evaluations have been compared with caffe-based implementations of standard neural networks that were deployed on a Titan-X GPU. Results showed that TrueNorth can detect cars from the dataset with 97.60% accuracy and can be used to accurately count the number of cars in the image with 69.04% accuracy. The car detection accuracy and car count (–/+ 2 error margin) accuracy are comparable to high-precision neural networks like AlexNet, GoogLeNet, and ResCeption, but show a manifold improvement in power consumption.

Keywords: deep learning, convolutional neural network, IBM TrueNorth Neurosynaptic System, neuromorphic computing, spiking neural network, aerial image analysis

OPEN ACCESS

Edited by:
Yansong Chua,
*Institute for Infocomm Research (A*STAR), Singapore*

Reviewed by:
Hesham Mostafa,
University of California, San Diego, United States
Roshan Gopalakrishnan,
*Institute for Infocomm Research (A*STAR), Singapore*

****Correspondence:***
Rohit Shukla
rshukla3@wisc.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *30 October 2018*
Accepted: *04 January 2019*
Published: *22 February 2019*

Citation:
Shukla R, Lipasti M, Van Essen B, Moody A and Maruyama N (2019) REMODEL: Rethinking Deep CNN Models to Detect and Count on a NeuroSynaptic System. Front. Neurosci. 13:4. doi: 10.3389/fnins.2019.00004

1. INTRODUCTION

Neural networks today are achieving state-of-the-art performance in competitions across a range of fields. Recent advances in deep learning (LeCun et al., 2015) have motivated the development of neural hardware substrates that are tailored to implementing deep networks with extremely low power and efficiency for a variety of embedded systems applications. Hardware that mimics the computational capabilities of a human brain through spiking neural networks has been shown to be not only extremely energy-efficient, but also capable of scaling up to large neural networks. Examples include the IBM TrueNorth Neurosynaptic System (Merolla et al., 2014), SpiNNaker (Furber et al., 2014), and the BrainScaleS project (Schemmel et al., 2008), all of which mimic the computational behavior of spiking neurons and can also be used to deploy deep neural networks.

One of the major challenges that these spiking neural network-based platforms faced was deploying convolutional neural networks (CNNs) on spiking neurons. This issue was addressed in the recent work from Cao et al. (2015) and Esser et al. (2016), and Eta Compute (Moore, 2018). The authors in Esser et al. (2016) have proposed an algorithm named energy-efficient deep neuromorphic networks (EEDN) to map CNNs on TrueNorth. EEDN networks achieved at or near state of the art accuracy when compared with traditional 32-bit precision neural networks on standard benchmarks and they operated at a much higher throughput (Frames Per Second) per watt. These promising results show potential for deploying spiking neural network based platforms for a variety of applications where battery life and power consumption are primary concerns. Such applications include video surveillance, UAV surveillance, aerial image analysis, etc.

Prior work such as Esser et al. (2015, 2016), Wen et al. (2016), Rueckauer et al. (2017), and Sengupta et al. (2018) have discussed about how to efficiently train neural network models so that the inference neural network can be easily mapped onto low precision hardware such as TrueNorth without any loss in output accuracy. But these prior works have only done the evaluations against small object recognition datasets such as MNIST, CIFAR-10, and CIFAR-100.

Prior work never listed out the challenges that might occur when mapping large CNN or DNN structures on TrueNorth for bigger datasets with large annotated images. For bigger datasets resource limitations and the CNN model limitations that TrueNorth can support start becoming a bottleneck. In this paper we evaluate the challenges related to deployment of EEDN trained neural network on TrueNorth hardware. Discussions that have been reported in this article are meant to complement the opportunities and challenges for spiking neural network hardware that have been reported in Pfeiffer and Pfeil (2018). The evaluations have been done against publicly-available dataset of overhead aerial images of cars that was proposed by Mundhenk et al. (2016) (Henceforth referred as COWC dataset). Examples from COWC dataset have been shown in **Figure 1**. As the neural network structures start becoming more complex, we have to keep in mind limited number of TrueNorth (Henceforth referred as TN) cores that are available and design a neural network structure so that we can obtain benefits by using hardware substrates more judiciously. This paper presents design decisions that a developer would have to make to design a neural network for the TrueNorth NS16e system (Sawada et al., 2016) that is shown in **Figure 2A**. The goal of this work is to present how knowledge of hardware architecture affects the decisions and parameter choices made while training and deploying neural networks on TrueNorth. These observations can assist us in maximizing the benefits of TrueNorth's available hardware computational resources.

Contributions of the research proposed in this paper are:

- Evaluate TrueNorth deployed CNNs for counting and detection tasks on COWC dataset (Mundhenk et al., 2016).
- Resources consumed by AlexNet (Krizhevsky et al., 2012) and VGG-16 (Simonyan and Zisserman, 2014) neural networks when deployed on NS16e hardware (Sawada et al., 2016). Identifying the architectural bottlenecks of these CNN structures and proposed changes to the CNN structure so that it could be deployed on NS16e hardware.
- Analysis of change in resource consumption and output accuracy based on the prior works such as, network-in-network structure (Lin et al., 2013), MobileNets (Howard et al., 2017), and YOLO (Redmon and Farhadi, 2016) neural network models.
- Discussions presented in section 4 outline the opportunities that are present in SNN hardware that can address the challenges present in TrueNorth architecture and EEDN training algorithm.

2. MATERIALS AND METHODS

2.1. Background

2.1.1. Cars Overhead With Context Dataset

Paraphrasing the work presented by Mundhenk et al. (2016), the cars overhead with context (COWC) data set is a large set of annotated overhead aerial images that contain cars. This dataset is useful for training Deep Neural Networks (DNNs) so that they are able to perform area based surveillance by detecting and counting cars that are present in the image. This dataset could be potentially used to keep track of volume of cars by deploying

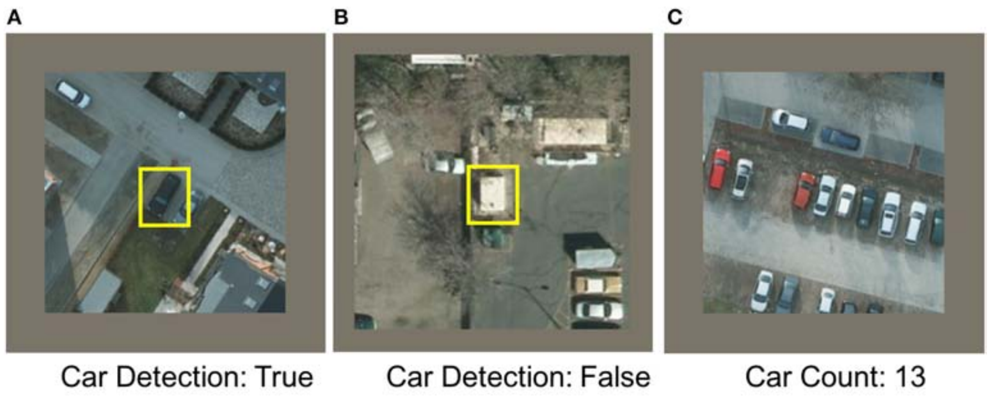


FIGURE 1 | Sample images from COWC dataset (Mundhenk et al., 2016). Images are 192-by-192 pixels. For detection, **(A,B)**, the model's goal is to detect whether a car is present in the center 48-by-48 pixels or not. Even though there are cars present in **(B)**, the label has been set to false because there is no car in the center 48-by-48 pixels of the image. For the counting task, **(C)**, the goal is to count the exact number of cars present in an image. The example shown in the figure has the label value "13," since there are 13 cars in the image.

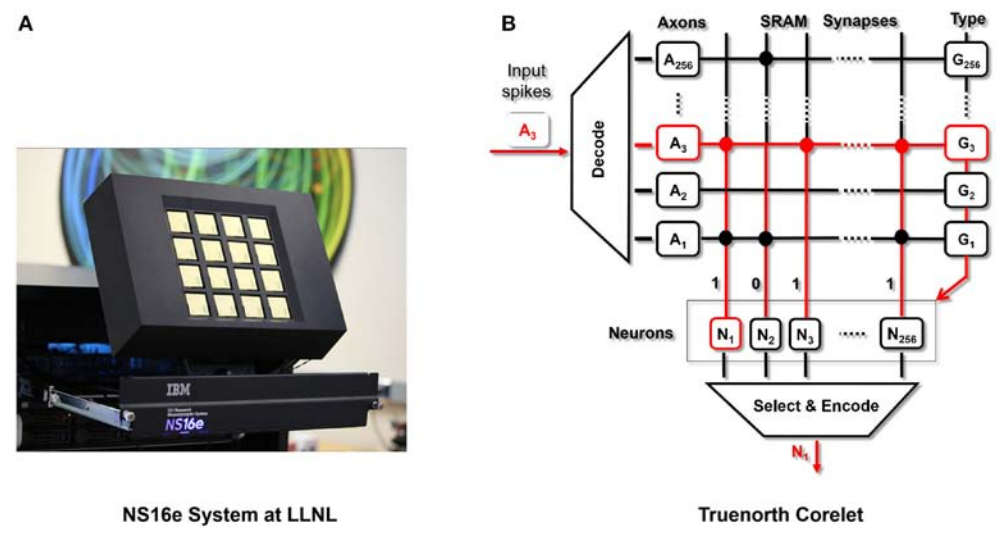


FIGURE 2 | (A) NS16e hardware system that was developed by IBM (Image from Shah, 2016). **(B)** Single neurosynaptic core which forms the computational block of the TrueNorth chips with the details presented in Cassidy et al. (2013) and Nere (2013).

the trained DNNs on unmanned aerial vehicles or drones. The goal of this dataset is to allow DNNs to determine the relationship between context and appearance such that something that looks very much like a car is detected even if it is in an unusual place. Unlike datasets such as MNIST, CIFAR-10, and CIFAR-100, where the maximum image size for which the neural network models were trained was 64-by-64 pixels (Esser et al., 2015, 2016), the COWC dataset consists of annotated images of size 192-by-192 pixels and this dataset requires us to solve a regression problem (counting the number of cars present in the entire image).

Figure 1 shows some of the sample images from the dataset. The goal of our work is to map this problem onto a low-power neural network architecture such as TrueNorth and evaluate its performance. The images in this dataset cannot be cropped out for training because the labels have been set for the entire image. For example, if the image shown in **Figure 1C** was cropped out for training, then the label "13" won't be correct, because the cropped out piece of image won't have the same number of cars as the label.

2.1.2. NS16e System

Summarizing the details of TrueNorth, as presented in Sawada et al. (2016), a single chip consists of 4,096 neurosynaptic cores (as shown in **Figure 2B**), tiled as a 64×64 array. Neurons integrate incoming spikes weighted by the synaptic strength and when a neuron membrane potential integrates beyond its threshold, it fires a spike, transmitting it to a target axon on any core in the network. In the same clock tick when neuron fired, the neuron would reset its membrane potential. Truenorth chips can be scaled beyond a single chip using SerDes links. As a result it is relatively simple to tile TrueNorth chips in a two-dimensional array, enabling the NS16e scale-up system.

Figure 3 shows a high-level setup for NS16e system and, the flow of computations happens between the off-chip system and NS16e hardware. In TrueNorth (as shown in **Figure 3A**) image binarization (data transduction) happens outside the TN chips, that is, in the CPU/FPGA hybrid system. ① When an RGB image is fed to the TrueNorth system, ② based on the learned convolutional layer weights and output feature count of the transduction layer, a corresponding number of binary images is produced. ③ These binary images are then sent to TN chips and on these TN chips these image features are fanned out using splitters (**Figure 3B**) so that multiple filter weights can operate in parallel on the same set of binary image features.

2.2. CNN Design Decisions

In this section we present design decisions for modifying standard neural network structures for NS16e hardware platform. First we will understand different set of computations that happen in standard neural network architectures, followed by what are the resource or architectural bottlenecks that we face when mapping these standard neural network architectures. Once we have understood the challenges and the architectural bottlenecks, we will look at how these issues can be addressed by proposing different neural network structure design.

2.2.1. Formulate Regression Problem as a Classification Problem

To maintain high throughput, TrueNorth performs operations in stream of single bits. A trained TrueNorth network will have ternary weights {−1,0,1} and binary activation {0,1}; as a result, algorithms that require us to solve regression problems, i.e., infer continuous output values, such as the car count in the image, present a challenge. Being able to estimate high precision values by using binary activation functions is a hard problem. In the context of TrueNorth and spiking neural networks, prior

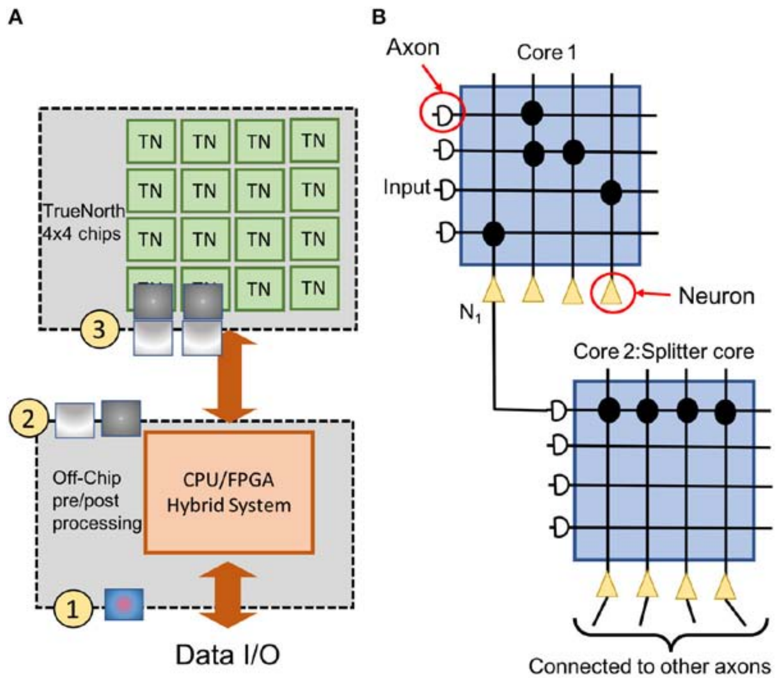


FIGURE 3 | The figure describes the NS16e system setup. **(A)** NS16e system consists of three stages. The hybrid CPU/FPGA system performs data pre/post processing and image binarization. The computed spikes are later sent to TN chips on which the CNN has been deployed **(B)** An image of how splitters are used on TrueNorth for increasing a neuron's fan-out.

work such as Diehl et al. (2016) and Shukla et al. (2017, 2018) have represented regression output values using rate coding scheme, where the expected value of spike train over a time window represented the value. But with this scheme the operating frequency of hardware starts becoming the bottleneck. To match the biological clock rate TrueNorth operates at 1 KHz frequency (Akopyan et al., 2015); as a result, if the problem requires us to estimate continuous numbers, we would have to count the number of spikes received over a window of time to estimate the output and this ends up slowing down the computation time. We can circumvent this issue by recasting the regression problem as a classification problem with estimated discrete values as outputs. This approach might require more hardware neurons for a large number of output bins. For the dataset that we are studying, the car counting problem would predict from 65 classes. As noted in Mundhenk et al. (2016), the number of cars in each image patch lie in the interval between 0 to 64.

2.2.2. Case Study: Map AlexNet Neural Network Model Onto TrueNorth

We will start off the discussion by mapping AlexNet neural network model onto TrueNorth NS16e hardware. The accuracy and hardware analysis of AlexNet-TrueNorth model has been presented in **Table 1**.

Figure 4A shows the neural network model of a standard AlexNet structure and **Figure 5A** shows the modified AlexNet neural network model for TrueNorth ns16e hardware. The difference between the neural network is highlighted using the rectangular box as described in **Figures 4B, 5B**. As shown in Esser et al. (2016), Equation (1) defines the activation function used by CNN layers that are deployed on TN.

TABLE 1 | Convolutional neural network structure analysis and testing accuracy.

Model name	Detection accuracy (in %)	Counting accuracy (in %)	Chips required for first 3 TN CNN layers
AlexNet (**Figure 4A**)	97.62	67.97	N/A
AlexNet modified (**Figure 5A**)	89.98	48.82	3.19
VGG-16 modified (1) (**Figure 8A**)	96.09	67.96	8.67
VGG-16 modified (2) (**Figure 8B**)	97.25	67.82	8.67
Deeper CNN structure 1 (**Figure 11A**)	97.52	68.21	11.16
Deeper CNN structure 2 (**Figure 11B**)	97.60	69.04	11.16

$$\text{TN defined activation function} = \begin{cases} 1 \text{ neuron filter response} \geq 0 \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

2.2.2.1. Challenges with AlexNet neural network model

In TrueNorth, neural network architectures where a large set of convolutional network neurons need to be connected to fully connected layers will consume a considerable amount

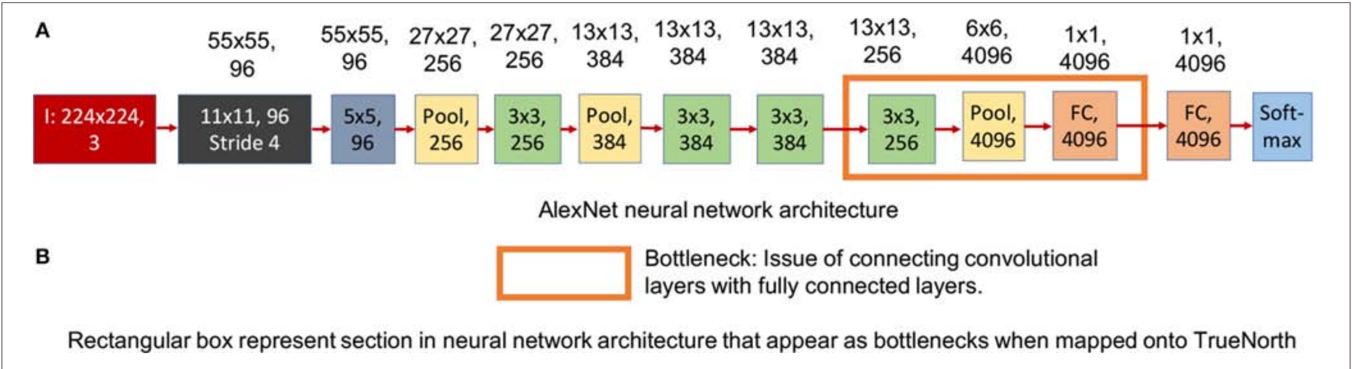

A
55x55, 96
55x55, 96
27x27, 256
27x27, 256
13x13, 384
13x13, 384
13x13, 384
13x13, 256
6x6, 4096
1x1, 4096
1x1, 4096
I: 224x224, 3
11x11, 96 Stride 4
5x5, 96
Pool, 256
3x3, 256
Pool, 384
3x3, 384
3x3, 384
3x3, 256
Pool, 4096
FC, 4096
FC, 4096
Soft-max
AlexNet neural network architecture
B
Bottleneck: Issue of connecting convolutional layers with fully connected layers.
Rectangular box represent section in neural network architecture that appear as bottlenecks when mapped onto TrueNorth

FIGURE 4 | This figure shows the standard AlexNet neural network architecture. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. **(A)** Shows the standard AlexNet neural network model (Krizhevsky et al., 2012). **(B)** Sections in the standard AlexNet neural network structure that pose a problem when trying to map it onto TrueNorth.

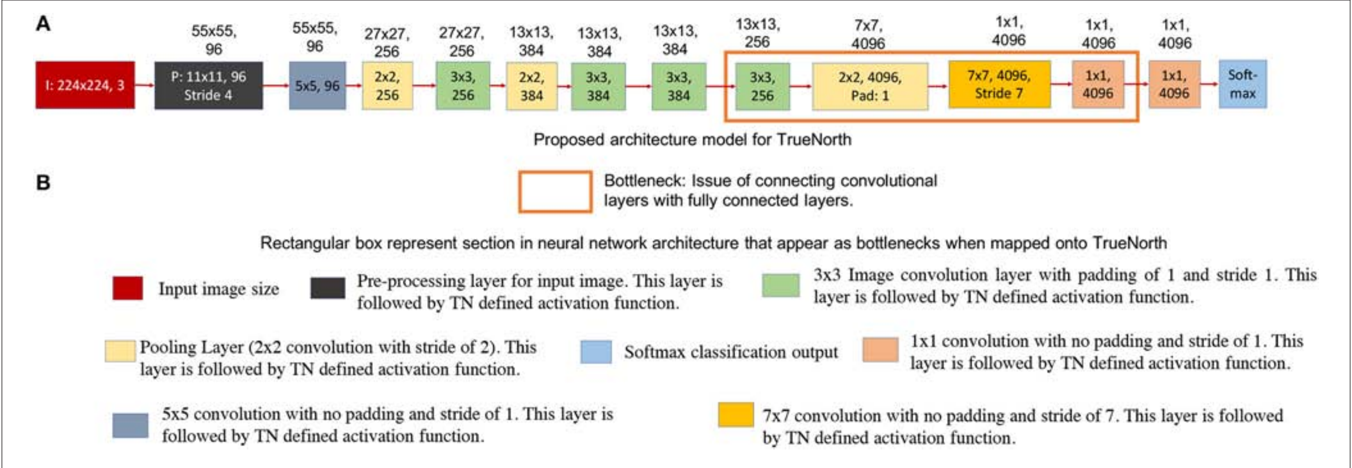

A
55x55, 96
55x55, 96
27x27, 256
27x27, 256
13x13, 384
13x13, 384
13x13, 384
13x13, 256
7x7, 4096
1x1, 4096
1x1, 4096
1x1, 4096
I: 224x224, 3
P: 11x11, 96 Stride 4
5x5, 96
2x2, 256
3x3, 256
2x2, 384
3x3, 384
3x3, 384
3x3, 256
2x2, 4096, Pad: 1
7x7, 4096, Stride 7
1x1, 4096
1x1, 4096
Soft-max
Proposed architecture model for TrueNorth
B
Bottleneck: Issue of connecting convolutional layers with fully connected layers.
Rectangular box represent section in neural network architecture that appear as bottlenecks when mapped onto TrueNorth
Input image size
Pre-processing layer for input image. This layer is followed by TN defined activation function.
3x3 Image convolution layer with padding of 1 and stride 1. This layer is followed by TN defined activation function.
Pooling Layer (2x2 convolution with stride of 2). This layer is followed by TN defined activation function.
Softmax classification output
1x1 convolution with no padding and stride of 1. This layer is followed by TN defined activation function.
5x5 convolution with no padding and stride of 1. This layer is followed by TN defined activation function.
7x7 convolution with no padding and stride of 7. This layer is followed by TN defined activation function.

FIGURE 5 | This figure shows the AlexNet implementation on TrueNorth. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. **(A)** Shows the modified AlexNet architecture for TrueNorth implementation. **(B)** Sections in the modified AlexNet neural network structure there later fixed when trying to map the standard AlexNet onto TrueNorth. The output feature dimensions of 9th CNN layer in the proposed modified AlexNet is different for standard AlexNet model (**Figure 4**). This is because the 8th CNN layer in this modified layer has a padding of 1, unlike the standard AlexNet mode where the 8th CNN layer did not have any padding.

of hardware resources. Thus, the proposed CNN avoid fully-connected layers, and instead the convolutional features are progressively downsampled to a one-by-one convolution. For example, in AlexNet (Krizhevsky et al., 2012), there are 9,216 neurons that present the output features of the 5th convolutional layer and these have to be connected to 4,096 neurons present in first fully connected layer. This kind of structure is crucial for datasets where we have to scan through the entire image pixels before predicting an output, such as counting the number of cars in our experiments. Prior work done by the authors have used either only a convolutional neural network structure (Esser et al., 2016) or just a fully connected neural network (Esser et al., 2015) in the context of object recognition. Earlier work have not addressed how to interface convolutional to fully connected layers. Mapping such CNN outputs on TrueNorth would require us to connect each convolutional layer neuron to all neurons in the fully connected layer. As a result, we might either end up using large number of cores as splitters to implement this fanout, as shown in **Figure 3B**, or we might use additional hardware resources to rearrange the 3D convolutional layers for a 1D fully connected layer.

2.2.2.2. Proposed modification for AlexNet neural network model

We have addressed the challenges associated with convolutional layer and fully connected layer connections by downsampling the CNN output all the way down to a one-by-one convolution using strided convolutions. The downsampling has been performed by having a convolutional layer that has convolution window of size 7 x 7 pixels and a stride of 7, as shown by the rectangular box in **Figure 5A**. Similar downsampling has been used in MobileNets (Howard et al., 2017). This structure ensures that the output layer considers the entire image but is more friendly to TrueNorth's limited fanout capability. The proposed AlexNet **Figure 5A** requires **9 TN chips** for deployment onto NS16e hardware.

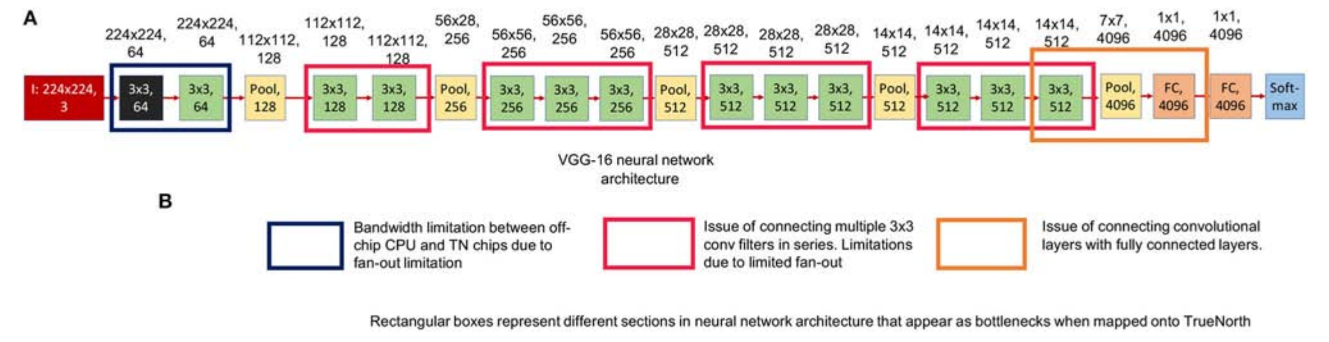

A
I: 224x224, 3
3x3, 64
Pool, 128
3x3, 128
Pool, 256
3x3, 256
Pool, 512
3x3, 512
Pool, 4096
FC, 4096
Soft-max
VGG-16 neural network architecture
B
Bandwidth limitation between off-chip CPU and TN chips due to fan-out limitation
Issue of connecting multiple 3x3 conv filters in series. Limitations due to limited fan-out
Issue of connecting convolutional layers with fully connected layers.
Rectangular boxes represent different sections in neural network architecture that appear as bottlenecks when mapped onto TrueNorth

FIGURE 6 | This figure shows the standard VGG-16 neural network architecture implementation. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. **(A)** Shows the standard VGG-16 neural network model (Simonyan and Zisserman, 2014). **(B)** Three sections in the standard VGG-16 neural network structure that pose a problem when trying to map it onto TrueNorth.

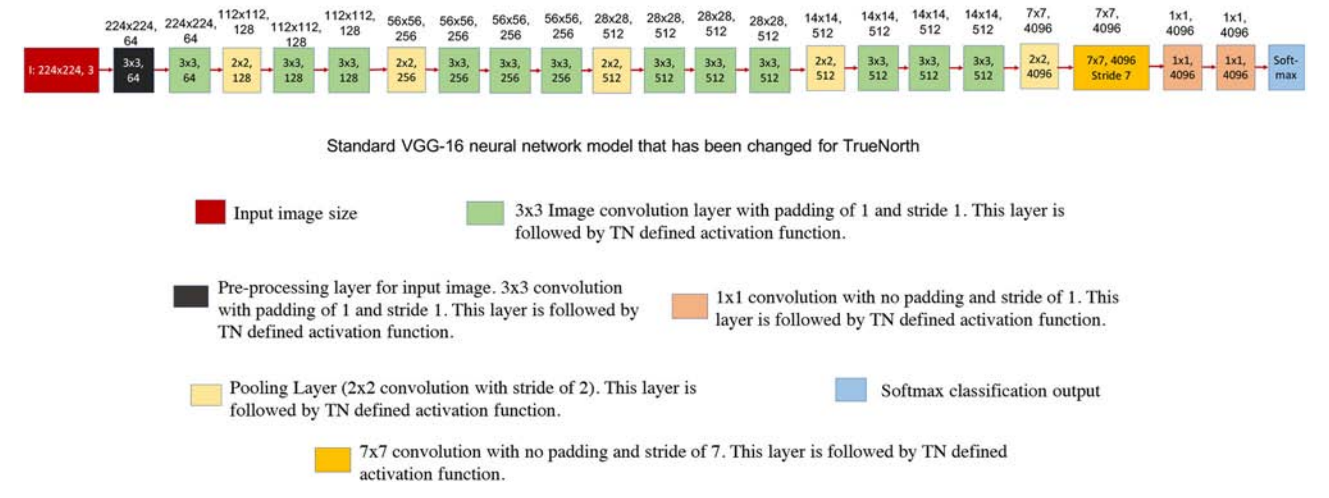

I: 224x224, 3
3x3, 64
2x2, 128
7x7, 4096 Stride 7
1x1, 4096
Soft-max
Standard VGG-16 neural network model that has been changed for TrueNorth
Input image size
3x3 Image convolution layer with padding of 1 and stride 1. This layer is followed by TN defined activation function.
Pre-processing layer for input image. 3x3 convolution with padding of 1 and stride 1. This layer is followed by TN defined activation function.
1x1 convolution with no padding and stride of 1. This layer is followed by TN defined activation function.
Pooling Layer (2x2 convolution with stride of 2). This layer is followed by TN defined activation function.
Softmax classification output
7x7 convolution with no padding and stride of 7. This layer is followed by TN defined activation function.

FIGURE 7 | This figure shows the standard VGG-16 neural network architecture that has been modified for TrueNorth implementation. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. Similar to AlexNet, this standard VGG-16 neural network model has CNN features that have been downsampled all the way down to a one-by-one convolution using convolution kernels of size 7 x 7 and stride of 7.

Readers should observe that the output feature dimensions of 9th CNN layer is different for standard AlexNet model (**Figure 4**) and modified AlexNet model (**Figure 5A**). This is because the 8th CNN layer in this modified layer has a padding of 1, unlike the standard AlexNet model where the 8th CNN layer did not have any padding.

2.2.3. Case Study: Map VGG-16 Neural Network Model Onto TrueNorth

Next we will look at the challenges that come up when we map VGG-16 style architecture onto the TrueNorth ns16e hardware. As explained earlier, Equation (1) defines the activation function used by CNN layers deployed on TN.

Figure 6A shows the neural network model of a standard VGG-16 structure. Three different sections of VGG-16 neural network structure that pose a problem for TrueNorth implementation have been highlighted using the rectangular box in **Figure 6B**.

Figure 7 shows the standard VGG-16 neural network architecture that has been modified for TrueNorth implementation. Similar to AlexNet, this standard VGG-16 neural network model has CNN features that have been downsampled all the way down to a one-by-one convolution using convolution kernels of size 7 x 7 and stride of 7.

2.2.3.1. Challenges in VGG-16: hardware resource limitation

If the users were to map the standard VGG-16 neural network model that has been shown in **Figure 7**, then the EEDN trained CNN model would require more than **49 TrueNorth chips** to deploy the said neural network; whereas, NS16e hardware has only 16 available TN chips. It is important for us to understand the architectural bottlenecks in the NS16e hardware that does not allow us to map the VGG-16 neural network structure and how can it be addressed when designing a neural network model for an application.

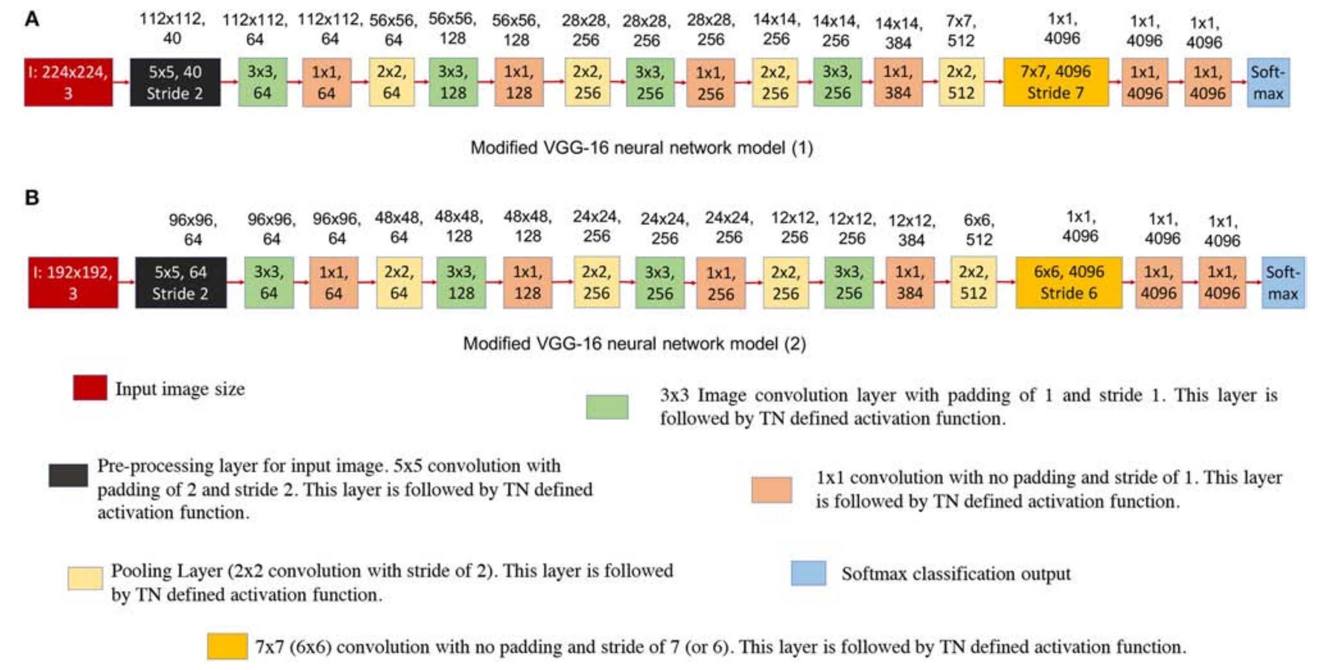

A
I: 224x224, 3
5x5, 40 Stride 2
112x112, 40
3x3, 64
112x112, 64
1x1, 64
112x112, 64
2x2, 64
56x56, 64
3x3, 128
56x56, 128
1x1, 128
56x56, 128
2x2, 256
28x28, 256
3x3, 256
28x28, 256
1x1, 256
28x28, 256
2x2, 256
14x14, 256
3x3, 256
14x14, 256
1x1, 384
14x14, 384
2x2, 512
7x7, 512
7x7, 4096 Stride 7
1x1, 4096
1x1, 4096
1x1, 4096
1x1, 4096
1x1, 4096
Soft-max
Modified VGG-16 neural network model (1)
B
I: 192x192, 3
5x5, 64 Stride 2
96x96, 64
3x3, 64
96x96, 64
1x1, 64
96x96, 64
2x2, 64
48x48, 64
3x3, 128
48x48, 128
1x1, 128
48x48, 128
2x2, 256
24x24, 256
3x3, 256
24x24, 256
1x1, 256
24x24, 256
2x2, 256
12x12, 256
3x3, 256
12x12, 256
1x1, 384
12x12, 384
2x2, 512
6x6, 512
6x6, 4096 Stride 6
1x1, 4096
1x1, 4096
1x1, 4096
1x1, 4096
1x1, 4096
Soft-max
Modified VGG-16 neural network model (2)
Input image size
3x3 Image convolution layer with padding of 1 and stride 1. This layer is followed by TN defined activation function.
Pre-processing layer for input image. 5x5 convolution with padding of 2 and stride 2. This layer is followed by TN defined activation function.
1x1 convolution with no padding and stride of 1. This layer is followed by TN defined activation function.
Pooling Layer (2x2 convolution with stride of 2). This layer is followed by TN defined activation function.
Softmax classification output
7x7 (6x6) convolution with no padding and stride of 7 (or 6). This layer is followed by TN defined activation function.

FIGURE 8 | This figure shows the modified VGG-16 neural network architecture for TrueNorth ns16e hardware. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. **(A)** Shows the modified VGG-16 neural network model (1) where the input image size if kept at 224x224 pixels. **(B)** Shows the modified VGG-16 neural network model (2) where the input image size if kept at 192x192 pixels.

2.2.3.2. Challenges in VGG-16: input feature size and feature count

In TrueNorth (as shown in **Figure 3A**) image binarization (data transduction) happens outside the TN chips, that is, in the CPU/FPGA hybrid system. As discussed in section 2.1.2, step 3, the binary image features representation are fanned out inside TN chips, thus, a considerable amount of resources are taken up by splitters for this pixel fan-out. That is, neurons that could have been potentially used for computation, have to be utilized as resources that would create multiple copies of the input features so that different convolutional filters can operate on these input features in parallel. Since prior work (Esser et al., 2015, 2016) have trained neural networks for a maximum input image size of 64-by-64 pixels, this problem of fan-out becomes more significant if the dataset has larger image size (192-by-192 pixels in case of COWC dataset). To minimize the fan-out resource utilization we have to either reduce the image size or reduce the number of input features. Next section will explain the reduction in required hardware resources for fan-out with the modified VGG-16 architecture (**Figure 8**). A more thorough analysis on the trade-off between fan-out requirement and, different input features and smaller input image sizes, has been presented in section 3.2.

2.2.3.3. Proposed modification for VGG-16 input

Figures 8A,B show the modified neural network models of VGG-16 structure and **Figure 9** shows the hardware requirements for mapping CNN layers on TrueNorth. For understanding the hardware resource consumption, we focus on the TN chips required by first three layers of CNNs deployed on TN and splitters. **Figure 8A** keeps the input image size same as the one for standard VGG-16 structure, but the number of features in the initial layer had to be reduced from 64 to 40. This is because having a feature count of 64 for the first layer requires 14 chips just to handle the fan-out using splitters. By reducing the number of feature count to 40, TrueNorth requires 3 chips for fan-out. Similarly, the fan-out constraints can be addressed by reducing the input image size as shown in **Figure 8B**. Here the goal was to keep the number of features in the initial layer to be 64, same as the one standard VGG-16 structure. To achieve this we have proposed an input image of comparatively smaller size, that is, instead of having an image of size 224 x 224 pixels, we have an input image of size 192 x 192 pixels. As explained earlier (section 2.1.1), the COWC dataset has images of size 192 x 192 pixels. Therefore, by having a comparatively smaller images as input we do not sacrifice any pixel level information, but after this modification we require only 5 TN chips to serve as splitters.

Figure 9 shows the breakdown of chip utilization for the splitters, and convolutional layers 2, 3, and 4, since these four layers consumed the most number of hardware resources. It can be inferred from **Figure 9** that by having small input feature size, TN requires significantly less number of hardware resources for splitters and the first CNN layer that is deployed on TN. AlexNet downsamples the input images by having a CNN layer of stride 5 in the initial layer. Whereas, for VGG-16 models, the user would have to keep in mind the input feature count and input image size because the initial layer has CNN layer of stride 2.

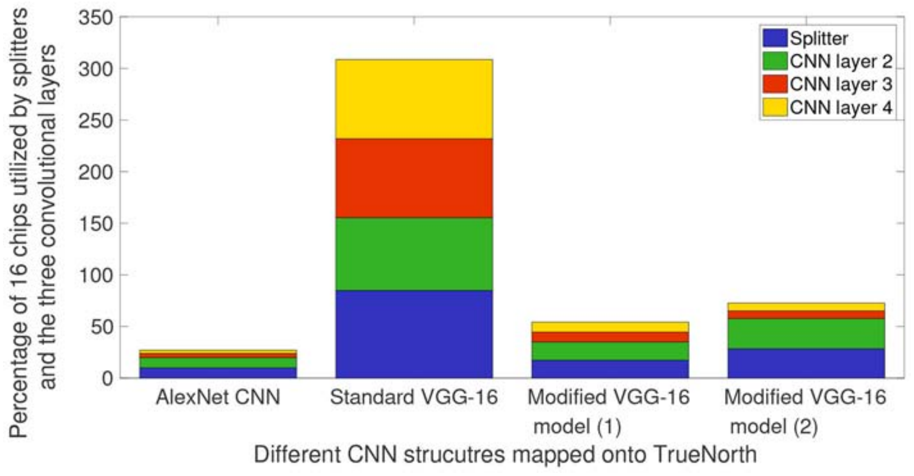


FIGURE 9 | Percentage of TN chips required on NS16e system for splitters and the three CNN layers that are deployed on the hardware. These chip consumption values are for AlexNet CNN presented in **Figure 5**, VGG-16 CNN models that have been presented in **Figures 7**, **8**.

2.2.3.4. Challenges in VGG-16: size of convolutional kernels

Selecting an appropriate convolutional kernel size is crucial for deploying CNNs on a hardware constrained substrate. Hence, smaller convolutional kernel would be very helpful. TrueNorth convolutional layers support 1 x 1 convolutions that were proposed by Lin et al. (2013). The pooling layers in EEDN networks have been implemented as convolutional operations with a stride of 2, as proposed by Springenberg et al. (2014). Larger kernels such as 5 x 5 kernels are good for learning higher level features in an image, whereas smaller kernels such as 3 x 3 and 1 x 1 kernels are good for learning lower level features and 1 x 1 convolutions can add non-linearity at a pixel level of the image. These convolution operations tend to learn the object properties and give prediction results based on these properties.

2.2.3.5. Proposed method for selecting kernel size

Convolution kernels that are bigger than 3 x 3, are used only in the preprocessing layers. As presented in section 2.1.2 and **Figure 3A**, image binarization or preprocessing happens off-chip. As a result, even if larger convolutional kernels are selected for the first CNN layer, TrueNorth resources do not get consumed because the first layer (or preprocessing layer) gets implemented off-chip. Therefore, as shown in **Figure 8**, the first CNN layer of modified VGG-16 structure has convolutional kernels of size 5 x 5 pixels and this layer is implemented off-chip. Similarly, we were able to have convolutional kernels of size 11 x 11 for the first CNN layer in modified AlexNet model as shown in **Figure 5A**. On the other hand, rest of the CNN layers have smaller sized convolutional kernels, that is, the convolutional kernels are of size 3 x 3 or 1 x 1. Smaller kernels require fewer computational resources, enabling us to fit a denser and wider network on the TrueNorth substrate. The 1 x 1 convolution layers require 9 times fewer groups than the 3 x 3 layers and 25 times fewer groups than the 5 x 5 layers. A similar idea of having only 1 x 1 and 3 x 3 convolution layers in the CNN structure was proposed by the authors of SqueezeNet (Iandola et al., 2016).

Figure 10 shows a comparison between hardware resources required by replacing certain 3 x 3 convolutions in standard VGG-16 neural network structure with 1 x 1 convolutions. Note that the x-axis of plot in **Figure 10** shows the CNN layer in standard VGG-16 that were replaced with 1 x 1 convolution kernels. 5th convolution layer of standard VGG-16 corresponds to 3rd convolution layer of modified VGG-16 structures; similarly 8th convolution layer of standard VGG-16 corresponds to 6th convolution layer of modified VGG-16 structures, 12th convolution layer of standard VGG-16 corresponds to 9th convolution layer of modified VGG-16 structures and 16th convolution layer of standard VGG-16 corresponds to 12th convolution layer of modified VGG-16 structures. It can be observed from the plots that by having smaller convolutional kernels, modified VGG model (1) (**Figure 8A**) is able to achieve up to 6.6x reduction in hardware resources; similarly modified VGG model (2) (**Figure 8B**) is able to achieve up to 8.3x hardware resources. Note that the second modified VGG model is performing computations on comparatively smaller image patches, as a result, it requires less number of hardware resources when compared with all of the other neural network structure models.

2.2.3.6. Discussion on fully convolutional neural network of VGG-16

As presented in section 2.2.2, one of the challenges that users might face when mapping standard neural network structures onto TrueNorth is that currently the proposed hardware architecture does not support convolutional layer to fully connected layer connections. Similar to modified AlexNet model, while mapping VGG-16 onto TrueNorth, the CNN features are downsampled all the way down to a one-by-one convolution using strided convolutions. The downsampling has been performed by having a convolutional layer that has convolution window of size 7 x 7 pixels and a stride of 7, (as shown in **Figure 8A**) or by having a convolutional layer that has

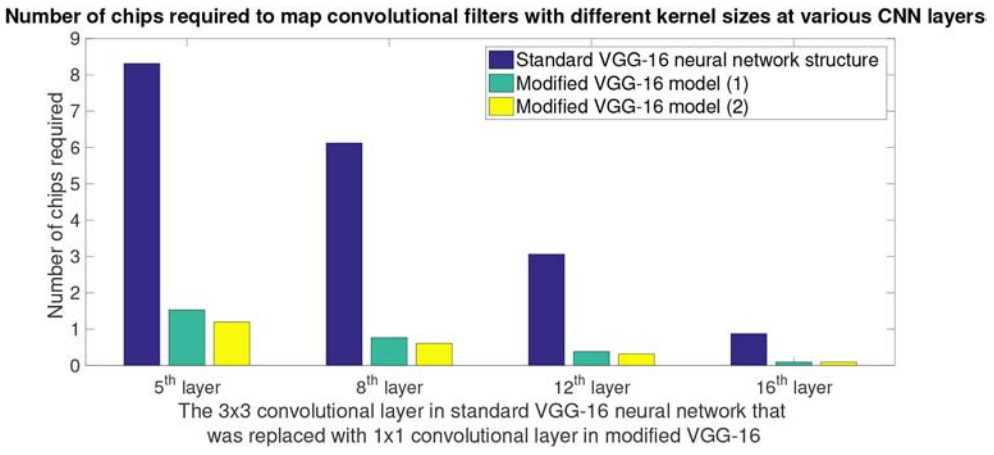


FIGURE 10 | Hardware savings that is achieved by replacing 3 x 3 convolution kernels in standard VGG-16 model with 1 x 1 convolution kernels. Modified VGG-16 model (1) refers to the CNN structure presented in **Figure 8A**, and modified VGG-16 model (2) refers to the CNN structure presented in **Figure 8B**. X-axis shows the convolutional layer in standard VGG-16 (Simonyan and Zisserman, 2014) CNN that originally had 3 x 3 convolution kernel, but they were replaced by 1 x 1 kernels in the modified VGG-16 (**Figure 8**) model for NS16e. Y-axis shows the number of chips that were consumed by the CNN layer when deployed onto NS16e.

convolution window of size 6 x 6 pixels and a stride of 6 (as shown in **Figure 8B**).

2.2.4. Case Study: Deeper Fully Convolutional Neural Network

As we have discussed in earlier designs, TrueNorth does not support convolutional layer to fully connected layer connections. The proposed solutions for the earlier neural network designs were to downsample intermediate CNN features all the way down to a one-by-one convolution using strided convolutions. We achieved this by taking average of CNN features that are of size 7 x 7 pixels (as shown in **Figures 5A**, **8A**) or 6 x 6 pixels (as shown in **Figure 8B**). In this section we propose having a deeper fully convolutional neural network for modified VGG-16 network (that were earlier shown in **Figure 8**). Unlike the proposed previous two designs, the CNN features are downsampled all the way down to a one-by-one convolution using additional strided convolutions of size 2 x 2 instead of having convolutional filters of size 7 x 7 or 6 x 6. The deeper convolutional neural network has been shown in **Figure 11**. The proposed deep CNN model does not require any additional TrueNorth chips for deployment. Since the image size has become significantly small, we do not observe any significant change in hardware requirements. As a result, the proposed deep CNN model can be mapped using all of the 16 TrueNorth chips that are available on NS16e hardware.

3. RESULTS

This section describes how the decisions that have been proposed in section 2.2 affect accuracy and hardware resource utilization. The EEDN-trained CNN structures have been compared against more standard neural network models that were deployed on Titan X GPU. All of the neural networks were trained only for COWC dataset. For EEDN trained CNNs the output layer has a *softmax* loss function. The car detection dataset had two output classes, whereas the car counting dataset has 65 output classes which predict car count from 0 to 64. Momentum was set at 0.9; the spikeDecay parameter which controls the backpressure of input spikes to a neuron was set at $7.5e-5$; and weightDecay parameter was set at $1e-6$ for all of the layers.

3.1. Accuracy Analysis

Table 2, shows the detection and accuracy for Alexnet (baseline neural network) and different CNN models that have been proposed in **Figures 5A**, **8A**, **8B**, **11A**, **11B**. The results of this table also quantifies the number of chips that are utilized to map the first three TN-deployed convolutional layers.

Based on the results reported in **Table 1**, a modified AlexNet model (**Figure 5A**) achieves significantly low accuracy compared to is floating-point counterpart (**Figure 4A**) that was implemented on a GPU. This loss in accuracy is due to ternary weight and binary activation representation that IBM TrueNorth computes on (as explained in McKinstry et al., 2018), as well as, aggressively downsampling the input images by a factor of 4 in the first layer because of which, the EEDN based CNN is not able to capture the unique features properly. Whereas, we can observe a significant improvement in accuracy with modified VGG-16 neural network models. Unlike AlexNet, the modified VGG-16 models (**Figures 8A,B**) are much deeper and are able to learn distinguishable features much more efficiently.

Figure 12 shows a comparison between counting labels estimated by AlexNet CNN structure (**Figure 5A**) and deep modified VGG-16 model (**Figure 12B**) that were deployed on TrueNorth. As stated earlier, AlexNet model is not able to learn the distinguishable features as efficiently as the deeper CNN models. It can be observed from the plots in **Figure 12** that average error is high for high value of counting labels. For high label values (45–49 and 50–54) images have high density of cars in them, therefore, it is important to have CNN structures that are able to learn the features which can detect individual cars and later use them for counting task.

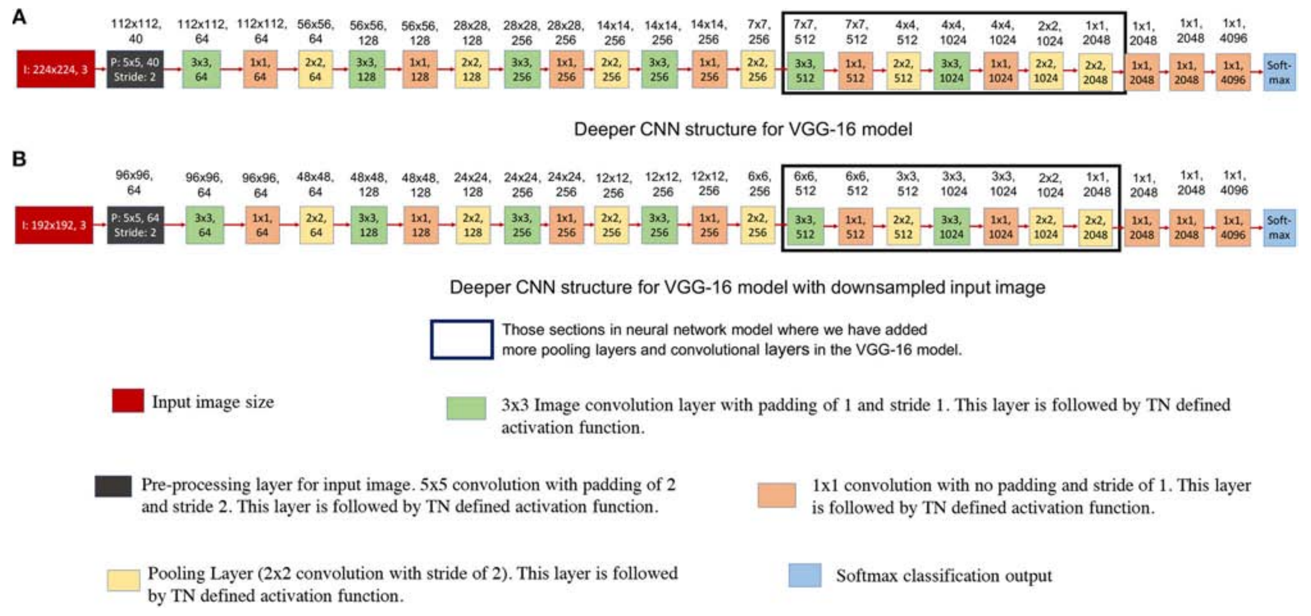


FIGURE 11 | This figure shows deeper convolutional neural network architecture for TrueNorth ns16e hardware. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. These CNN models are extensions of the VGG-16 models that were proposed in **Figure 8**. **(A)** shows the deep convolutional neural network model where the input image size is kept at 224x224 pixels. **(B)** shows the deep convolutional neural neural network model where the input image size is kept at 192x192 pixels.

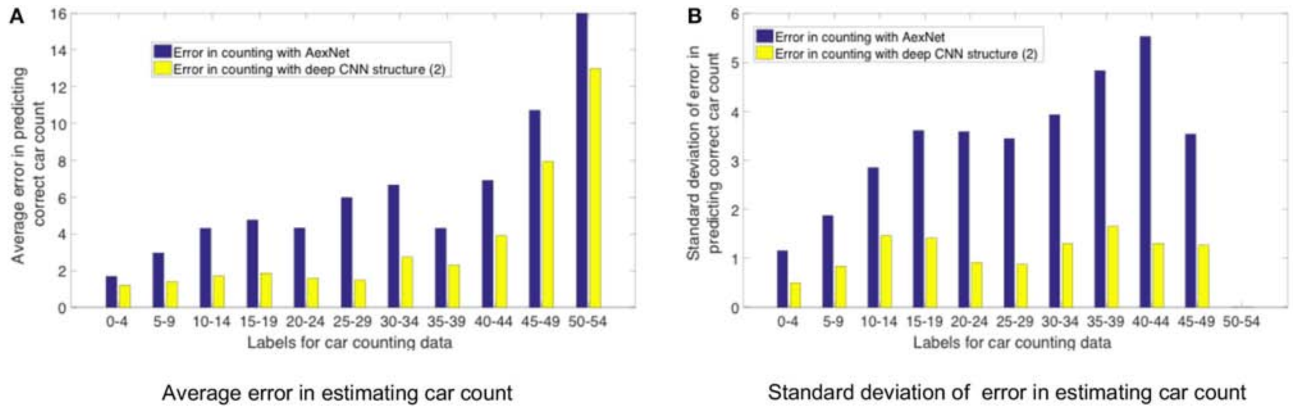


FIGURE 12 | Error in estimating the label of car count vs the actual car count label. The plot compares the counting labels that were predicted with AlexNet CNN (**Figure 5A**) and deep modified VGG-16 model (**Figure 11B**). X-axis shows the range of labels associated with the counting dataset. For example, in the x-axis a value of 0–9 represents all of the counting dataset labels that were counting values in the range from 0 to 9. In **(A)** Y-axis plots the average error in estimating car count, and in **(B)** Y-axis plots the standard deviation of error in estimating car count.

Table 2, shows the detection and accuracy for Alexnet and different CNN models that have been proposed in **Figure 13**. The results of this table also quantifies the number of chips that are utilized to map the first three TN-deployed convolutional layers.

3.2. Experiments With Additional Neural Network Structures

Figure 13 shows the different CNN models that were trained using EEDN training algorithm. All of these proposed CNN models are a variation of deep CNN structure that was shown in **Figure 11**. Equation (1) shows the activation function used by CNN layers deployed on TN. It is important for us to understand how different input image size or feature count of convolutional layers would affect the hardware resource consumption and the test accuracy. If the CNN structure is designed naively, then we might waste critical compute resources for performing operations such as creating multiple instances of input data. On the other hand, if the proposed design is extremely conservative, then the accuracy may reduce significantly. Therefore, in this section we will discuss how different design proposals will affect hardware usage and dataset accuracy.

TABLE 2 | Hardware resource analysis and testing accuracy for additional CNN structures.

Model name	Detection accuracy (in %)	Counting accuracy (in %)	Chips required for first 3 TN CNN layers
AlexNet	97.62	67.97	N/A
CNN Model 1 (**Figure 13A**)	97.87	68.62	19.88
CNN Model 2 (**Figure 13B**)	97.21	66.73	9.45
CNN Model 3 (**Figure 13C**)	97.52	68.21	8.67
CNN Model 4α (**Figure 13D**)	97.60	69.04	11.16
CNN Model 4β (**Figure 13E**)	90.98	53.4	11.16
CNN Model 5 (**Figure 13F**)	97.1	65.31	8.99

Each of the proposed CNN structure has a different input image size, and different output feature counts for the first four convolutional layers. The first convolutional layer (or transduction layer) is deployed on CPU/FPGA off-chip system (Esser et al., 2016; Sawada et al., 2016), whereas convolutional layers 2, 3, and 4 are deployed on the TrueNorth hardware. The proposed CNN models, **Figures 13B–F** require 16 chips to be deployed on TN hardware.

The CNN models shown in **Figures 13A–D,F**, are all 23-layered CNN models, and the final layer serves as softmax loss function. **Figures 13D,E** are meant for comparison with prior approach to model CNNs. **Figure 13E** is a 19-layered CNN model, and in this structure we do not downsample the image features to a 1 × 1 patch. Instead for CNN model 4β (**Figure 13E**) we downsample the patches until the size of the patch is 6-by-6 pixels. Even though CNN models 4α (**Figure 13D**) and 4β (**Figure 13E**) have a different number of layers, the input image size and feature count in the initial layers are the same for the both models

Figure 14 shows the breakdown of chip utilization for the splitters, and convolutional layers 2, 3, and 4, since these four layers consumed the most number of hardware resources. In section 2.2.3.2 we introduced the concept of balancing input image size with the transduction layer's output feature count so that a minimum number of chips are used up for fan-out while keeping the test accuracy comparable to more standard approaches. **Table 2** shows that by proposing a neural network architecture that is similar to CNN model 4, we can have test accuracy that is similar to the full precision AlexNet implementation. In CNN model 4 (**Figure 13C**), the input image is of size 192-by-192 pixels, as a result, there is no loss in pixel information due to early downsampling. If input images are downsampled aggressively (by using pooling layers), or the number of features is reduced significantly, test accuracy for detection and counting will also decrease. For example, if the input images are downsampled from 160-by-160 pixels to a small size of say 80-by-80 pixels in the first convolutional layer, then we can have more number of features, but the output accuracy is still less compared to CNN model 4. Having more output feature does not help in improving the test accuracy because the image features do not get captured nicely with an aggressive downsampling operation.

3.3. Comparison With Prior Approach

Section 2.2.3.6 motivated the need for fully convolutional neural networks where the image patch has been downsampled to a 1 × 1 patch. Prior work by Esser et al. (2016) proposed a fully convolutional neural network where a 64-by-64 pixel input image was downsampled to an 8-by-8 patch for output prediction. We compare our proposed CNN structure with the decision that was presented (Esser et al., 2016) and (Alom et al., 2018). We perform this comparison by analyzing the test accuracy of CNN model 4α (**Figure 13D**) and CNN model 4β (**Figure 13E**). In CNN model 4β, the input image patch is downsampled only to a 6-by-6 pixel patch. Both of these CNN models require 16 TN chips to be deployed. The training parameters were also the same for both of these models.

Based on the results shown in **Table 2**, we can observe there is a significant difference in test accuracy between the two models. This might be because CNN model 4β does not get to scan the entire image before making the prediction. In contrast, CNN model 4α is able to find a relationship between all of the pixels in the image and provide a better output prediction. There is a difference of 6.62% in detection accuracy and 15.64% in counting accuracy between CNN model 4α and CNN model 4β, with our approach of CNN model 4α having a considerably higher test accuracy.

3.4. Hardware Analysis

As per the detection and counting accuracies shown in **Table 2**, CNN model 4α (**Figure 13C**) has the best accuracy among all of the neural network models that were evaluated. This model can also be deployed on NS16e TrueNorth hardware. Therefore, rest of the discussion in this section will focus on the test accuracy results obtained from CNN model 4, as well as report hardware analysis for this neural network model.

Table 2 shows the results for COWC dataset after the trained network (CNN model 4) was deployed on NS16e system. Neural network structures for both counting and detection tasks consumed all of the 16 chips available in NS16e platform. The standard neural networks were implemented using the Caffe neural network framework (Jia et al., 2014) and the trained full-precision neural networks were deployed on NVIDIA Titan X GPU. **Table 3** shows the percentage accuracy for three different tasks. The first task is car detection, a binary classification problem where the goal is to predict whether a car has been detected in the center of the image or not. For the entire detection test dataset, accuracy of car detection with CNN model 4 (**Figure 13C**) is 97.35%, precision score is 96.36%, recall score is 97.33%, and the F1 score of this task is 96.84%. Overall, the mapped neural network on TrueNorth does very well in detecting the objects. The second task is to count the number of cars in the image and predict how many cars are present in the image in the range from 0 to 64. The third goal is to count the number of cars in the image by relaxing the output prediction condition; that is, if an error margin of $-/+ 2$ is allowed for estimating the car count, then what would be the prediction accuracy. For example, in **Figure 1C** the correct label is 13 for counting. With $-/+ 2$ margin error, if the neural network predicts any label in the range

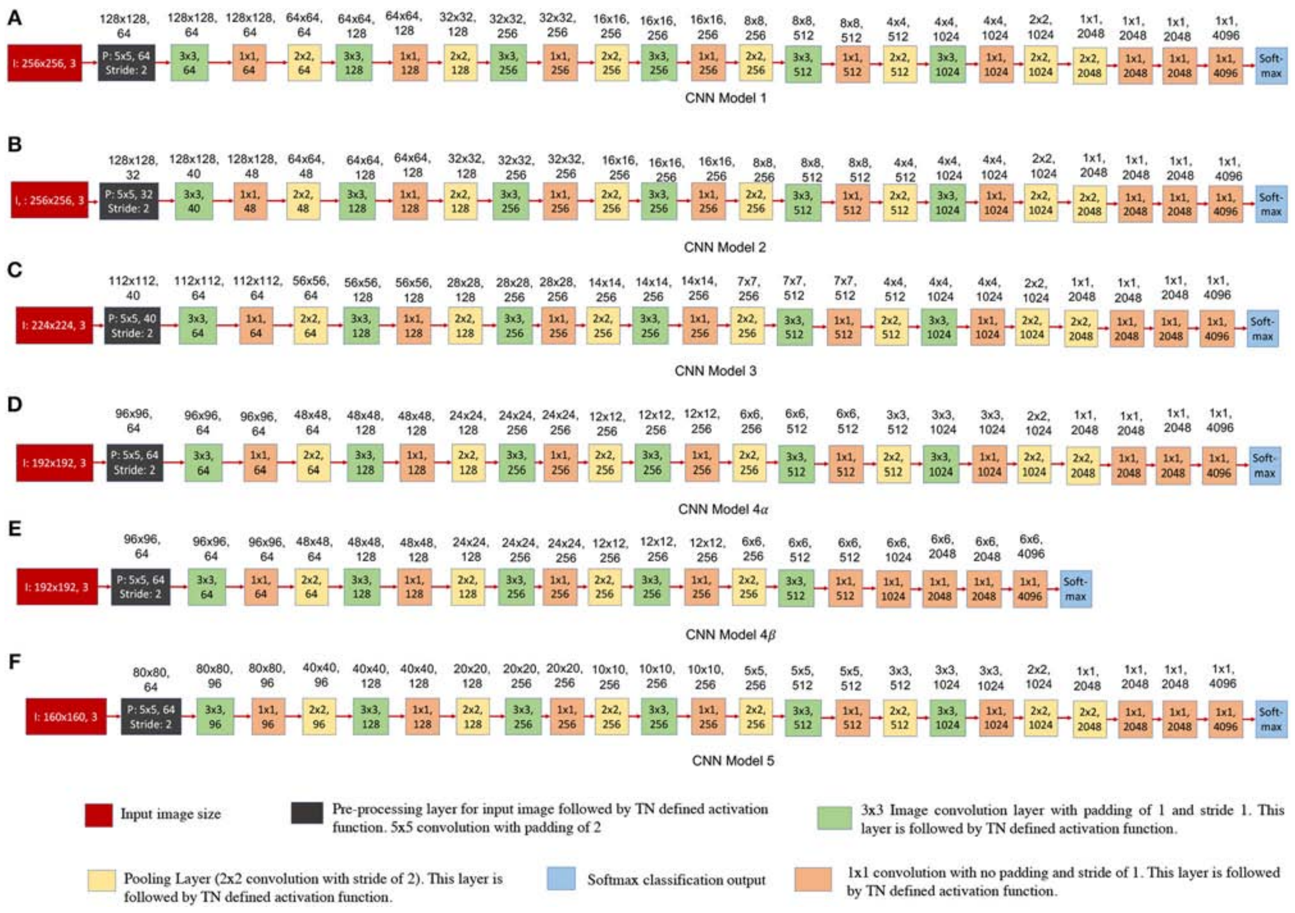

CNN Model 1
CNN Model 2
CNN Model 3
CNN Model 4α
CNN Model 4β
CNN Model 5
Input image size
Pre-processing layer for input image followed by TN defined activation function. 5x5 convolution with padding of 2
3x3 Image convolution layer with padding of 1 and stride 1. This layer is followed by TN defined activation function.
Pooling Layer (2x2 convolution with stride of 2). This layer is followed by TN defined activation function.
Softmax classification output
1x1 convolution with no padding and stride of 1. This layer is followed by TN defined activation function.

FIGURE 13 | Convolutional neural network structures trained using EEDN for COWC dataset. The numbers written on top of the blocks show the output feature dimension of that block in CNN model. **(A–F)** shows different design decisions for all of the six CNN models. Each of the proposed CNN model either has (1) different input image size, or (2) different output feature count for first four convolutional layers, or (3) different number of pooling layers (CNN models 4α and 4β). **(A–D)** and **(F)** are all 23-layered CNN models, and the final layer serves as softmax loss function. **(D)** and **(E)** are meant for comparison with prior approach to model CNNs. **(E)** is a 19-layered CNN model, and in this structure we do not downsample the image features to a 1 × 1 patch.

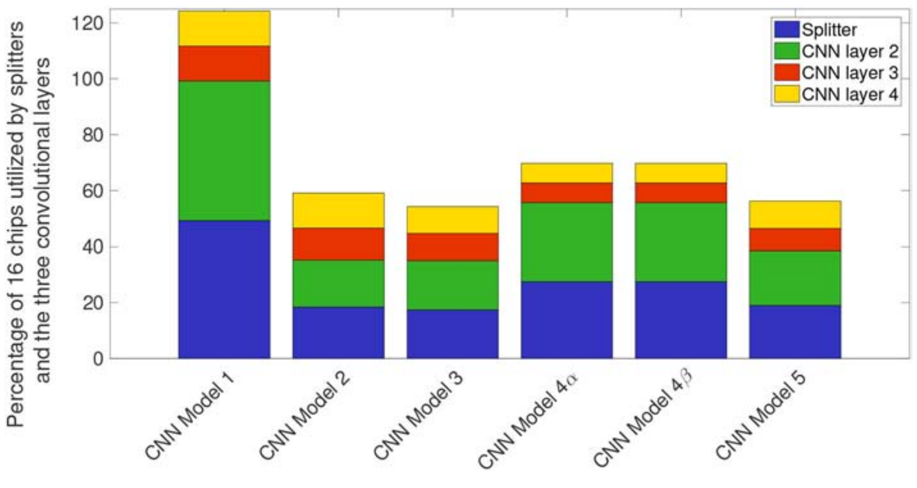


FIGURE 14 | Stacked bar plot illustrating percentage chips utilized in the NS16e system by splitters and convolutional layers for different convolutional models.

TABLE 3 | This table reports accuracy for car detection and car counting on TrueNorth and NVIDIA Titan X, as well as throughput for car counting on these platforms.

Neural network structure	Detection accuracy (in %)	Counting accuracy (in %)	Counting accuracy with −/+2 error margin (in %)	Frames per second (FPS) for counting task	FPS per watt	
Truenorth (EEDN)	97.60	69.04	96.57	3.38	0.444 (at 0.775 V)	0.387 (at 1.0 V)
AlexNet	97.62	67.97	98.82	11.48	0.046	
GoogLeNet	99.12	80.35	98.87	2.73	0.011	
ResCeption	99.14	80.34	98.86	2.91	0.012	

∈ [11, 15] our model would classify that as a correct output with respect to the input image of **Figure 1C**.

As per the results for CNN model 2 (**Figure 13C**) in **Table 3**, neural networks deployed on TrueNorth with EEDN framework have accuracy that is close to AlexNet, but have a considerable difference when compared with GoogLeNet (as proposed in Szegedy et al., 2014) and ResCeption (as proposed in Mundhenk et al., 2016). This could be due to the rich feature representations that GoogLeNet and ResCeption can capture. Each layer in these two neural networks has different-sized filters operating in parallel, and the outputs from these filters get depth concatenated. As a result GoogLeNet and ResCeption can capture robust, differentiable features. However, this difference reduces significantly with an allowed error margin of −/+ 2 when predicting the car count.

Table 3 shows the frames per Second (FPS) for car counting based classification problem for the neural networks that were deployed on different hardware platforms. As per (Mundhenk et al., 2016) a single frame in FPS is defined as the scene of size 2048-by-2048 pixels with additional padding so that the first patch has a center at (0,0). The image frame is divided into multiple patches of 192-by-192 pixels and a stride of 167 pixels. Therefore, the frames per Second (FPS) for counting task is meant to quantify how fast the CNN models can scan though an entire image frame of 2048-by-2048 pixel and be able to count the number of cars in this entire frame. The tick period of TrueNorth operation had to be increased to 1.75 ms (operating frequency was reduced to 571.43 Hz) to get the results shown in **Table 3**, possibly because for smaller tick period, spikes were getting bottlenecked when trying to cross chip boundaries. Article on TrueNorth ecosystem (Sawada et al., 2016) presents how spikes travel during inter-chip communication. First a spike has to traverse one row of the network-on-chip, then travel through the chip I/O peripheral circuitry and finally it is delivered to the destination chip through limited I/O connections that are present between two chips. Since the spikes have to travel peripheral circuitry and limited I/O connections that are present between two chips, these sections become a bottleneck for inter-chip communication if the spike rate is high. As a result, the spikes were not getting delivered for smaller tick periods since the inter-chip communication bandwidth was becoming the bottleneck for multi-chip networks. Prior work by Akopyan et al. (2015) have proposed wire-length minimization placement algorithm for TrueNorth. A better placement of cores could improve the runtime as well as the FPS.

In this section we report the first-order analysis of NS16e TrueNorth power consumption values based on the analysis that was presented in Merolla et al. (2014) and Sawada et al. (2016). TrueNorth chips can operate at 0.775 V and 1.0 V. The power consumption values were calculated with an operating frequency of 571.43 Hz, static power was set to 70 mW for 0.775 V operating voltage and 114 mW for 1.0 V operating voltage. We assumed that dynamic power is the same as static power for an operating frequency 1KHz and later these dynamic power values were scaled down linearly to account for the chip operating frequency of 571.43 Hz. When all of the chips on NS16e board are computing at the same time, the total combined active power consumed by TrueNorth chips is 1.76 W and 2.87 W with the operating voltage set at 0.775 V and 1.0 V, respectively. Total peak power consumed by the NS16e system is 7.62 W for 0.775 V operating voltage and 8.73 W for 1.0 V operating voltage. In contrast, an NVIDIA Titan X GPU can consume a peak power of 250 W to run these neural network structures at its highest frames per second rate.

4. DISCUSSION

4.1. Summary

In this paper we described four design decisions that a designer would have to address to deploy CNN structures on a neurosynaptic system such as IBM TrueNorth. These decisions are very important if the goal is to perform tasks such as detection and counting in a hardware constrained environment. Section 2.2 introduced the need to have a systematic approach for proposing neural network designs that can be mapped onto TrueNorth. Here we discussed how we can leverage prior work that have been proposed for CNN design and extend those ideas to EEDN based CNN models for TrueNorth. We showed that if a standard VGG-16 CNN model is modified systematically, while keeping in mind the architectural bottlenecks that are present in NS16e, hardware resource requirements can be reduced by 3x (refer to **Figures 9**, **10**).

Similarly, we discussed in **Table 1** that with systematic approach to mapping CNNs on TrueNorth, the accuracy could be improved by 8% for detection based task and by 20% for counting based task when compared to having a naive ternary-weight AlexNet implementation on NS16e. Results presented in **Table 2** show that EEDN trained neural network can have similar accuracy as full precision AlexNet.

It is important for us to consider how many TN cores are performing relevant computations. The analysis presented in **Figure 14** shows that it is extremely important for users to consider the trade-off between the hardware resources that is available for mapping the neural network, and the input image size and feature counts of initial layers, to achieve the desired test accuracy.

Section 3.4 analyzes the cost of the deployed neural network on TN hardware. As per the results presented in **Table 3**, the EEDN-trained neural network when deployed on TN hardware has test accuracy that is comparable to high-precision neural networks like AlexNet, GoogLeNet, and ResCeption, but shows a manifold improvement in FPS per watt.

4.2. Extending This Work to Other Benchmarks and Neuromorphic Chips

As neuromorphic computing is becoming more promising, it is important for researchers to understand the challenges that came up in TrueNorth architecture/algorithm and address these issues in future neuromorphic computing architectures/algorithms.

First, it is important for us to have a new set of benchmarks and datasets that can be used to evaluate neuromorphic hardware for bigger CNN models or that require us to estimate continuous numbers such as regression problems. There have been benchmarks that were proposed keeping in mind SNN algorithms, viz., N-MNIST (Orchard et al., 2015) and CIFAR-10 DVS (Li et al., 2017), but both of these benchmarks have very small image sizes and both of these benchmarks can solved using classification models. Problems that require us to estimate continuous numbers bring out the architectural limitations that might arise if the goal is to predict large range of numbers. On the other hand, benchmarks from domains such as Micro-Aerial Vehicles (Ma et al., 2013) and video surveillance would be very interesting for the SNN community because these small drones already have SNN controllers in them (Clawson et al., 2016). Having video surveillance dataset from MAVs, will help us realize potential of SNNs to be deployed in energy-constrained environments. Evaluating the hardware with bigger CNN models will help us understand the architectural limitations that are present in the hardware and it will also motivate researchers to investigate better algorithms for hardware/software co-design on neural networks.

Second, it is critical to investigate the fan-out limitations of architectures such as TrueNorth, so that neural networks can also support connections between convolutional and fully-connected layers. Even though there have been prior research that have proposed algorithms to train inception neural networks or residual networks for SNN hardware (Rueckauer et al., 2017; Sengupta et al., 2018), the current architectural limitations related to fan-out in SNN hardware such as TrueNorth, do not support such skip connection based CNNs. Concurrently, CNN structures such as MobileNets (Howard et al., 2017) have shown to significantly reduce the memory accesses and computations for embedded platforms. To the best of author's knowledge, currently there is no research that has successfully trained ternary quantized model for depthwise separable filters, which is a critical part of MobileNets. Prior work done in Holesovsky and Maki (2018) have attempted to train a depthwise separable CNN with ternary weights and activation, but reported a significant drop in accuracy when compared to the same CNN structure that was trained with single precision weights and activation.

Third, it is important to address the architecture bottlenecks present between the CPU/FPGA hybrid system and the neuromorphic chips, otherwise, a considerable amount of computation resources may end up getting used up to handle these interactions, as shown in CNN baseline example of **Figure 14**. Another direction that researchers can potentially investigate is improving the speed of deployed neural networks by analyzing the bottleneck present during inter-chip communication on a scaled-up hardware such as NS16e system.

Finally, as neural network models become deeper and wider, there will be a considerable amount of communication happening between neurons mapped onto different chips. This bottleneck could be addressed by having a better placement algorithm for multi-chip placement which would constrain group neurons that communicate a lot with each other to a single chip, unlike the work proposed in Akopyan et al. (2015) where the goal of the placement algorithm is to minimize the wire-length of placed neurons. Or, researchers can propose a new interconnect architecture for inter-chip communication that could handle high backpressure of spikes that get delivered from one neuromorphic chip to another.

Pruning may not always be the best approach to address hardware constraints while DNN training. As presented in Yazdani et al. (2018) even though pruning may give correct test accuracy, the inference confidence score reduces significantly. Researchers from hardware community have proposed pruning algorithms to reduce the size of bigger CNNs for hardware deployment (Han et al., 2015; Iandola et al., 2016). At present EEDN trained CNN models are highly sparse due to ternary weight representation, having more aggressive, such as pruning away TN cores for deep learning model, pruning technique may result in further drop in test accuracy. Therefore, rethinking the placement strategy for deep learning models on SNN may be an important step forward to address the issue of hardware constraints.

AUTHOR CONTRIBUTIONS

RS was the one that led this project. He came up with the idea, suggested the plan of execution, performed all of the experiments and wrote this paper. ML, BV, AM, and NM provided feedback for the work that RS did and also gave suggestions about how to improve the manuscript.

ACKNOWLEDGMENTS

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-JRNL-767281). Experiments were performed at the Livermore Computing facility.

REFERENCES

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Alom, M. Z., Josue, T., Rahman, M. N., Mitchell, W., Yakopcic, C., and Taha, T. M. (2018). "Deep versus wide convolutional neural networks for object recognition on neuromorphic system," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro), 1–8. doi: 10.1109/IJCNN.2018.8489635

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vision* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. (2013). "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–10. doi: 10.1109/IJCNN.2013.6707077

Clawson, T. S., Ferrari, S., Fuller, S. B., and Wood, R. J. (2016). "Spiking neural network (SNN) control of a flapping insect-scale robot," in *2016 IEEE 55th Conference on Decision and Control (CDC)* (Las Vegas, NV), 3381–3388. doi: 10.1109/CDC.2016.7798778

Diehl, P. U., Pedroni, B. U., Cassidy, A., Merolla, P., Neftci, E., and Zarrella, G. (2016). "TrueHappiness: neuromorphic emotion recognition on TrueNorth," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC), 4278–4285. doi: 10.1109/IJCNN.2016.7727758

Esser, S. K., Appuswamy, R., Merolla, P. A., Arthur, J. V., and Modha, D. S. (2015). "Backpropagation for energy-efficient neuromorphic computing," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15 (Cambridge, MA: MIT Press), 1117–1125.

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: compressing deep neural network with pruning, trained quantization and Huffman coding. *CoRR abs/1510.00149*.

Holesovsky, O., and Maki, A. (2018). "Compact ConvNets with ternary weights and binary activations," in *23rd Computer Vision Winter Workshop* (Cesky Krumlov).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv[Preprint].arXiv:1704.04861*.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Han, S., Dally, J., et al. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *arXiv[Preprint].arXiv:1602.07360*.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv[Preprint].arXiv:1408.5093*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25,* eds F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Lake Tahoe, NV: Curran Associates, Inc.), 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci.* 11:309. doi: 10.3389/fnins.2017.00309

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

Ma, K. Y., Chirarattananon, P., Fuller, S. B., and Wood, R. J. (2013). Controlled flight of a biologically inspired, insect-scale robot. *Science* 340, 603–607. doi: 10.1126/science.1231806

McKinstry, J. L., Esser, S. K., Appuswamy, R., Bablani, D., Arthur, J. V., Yildiz, I. B., et al. (2018). Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv[Preprint].arXiv:1809.04191*.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada,J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Moore, S. K. (2018). *Eta Compute Debuts Spiking Neural Network Chip for Edge AI.* IEEE spectrum. Available online at: https://spectrum.ieee.org/tech-talk/semiconductors/processors/eta-compute-debuts-spiking-neural-network-chip-for-edge-ai

Mundhenk, T. N., Konjevod, G., Sakla, W. A., and Boakye, K. (2016). "A large contextual dataset for classification, detection and counting of cars with deep learning," in *14th European Conference on Computer Vision* (Amsterdam).

Nere, A. (2013). *Computing with Hierarchical Attractors of Spiking Neurons*. PhD thesis, University of Wisconsin - Madison.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Redmon, J., and Farhadi, A. (2016). Yolo9000: Better, faster, stronger. *arXiv[Preprint].arXiv:1612.08242*.

Rueckauer, B., Lungu, I. A., Hu, Y., Pfeiffer, M., and Liu, S. C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., et al. (2016). "TrueNorth ecosystem for brain-inspired computing: scalable systems, software, and applications," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT), 130–141.

Schemmel, J., Fieres, J., and Meier, K. (2008). "Wafer-scale integration of analog neural networks," in *Proceedings of the International Joint Conference on Neural Networks* (Hong Kong), 431–438.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2018). Going deeper in spiking neural networks: VGG and residual architectures. *arXiv[Preprint].arXiv:1802.02627*.

Shah, A. (2016). *Ibm's Brain-mimicking Computers are Getting Bigger Brains*. Available online at: https://www.pcworld.com/article/3050444/hardware/ibm-is-creating-larger-brain-mimicking-computers.html

Shukla, R., Jorgensen, E., and Lipasti, M. (2017). "Evaluating hopfield-network-based linear solvers for hardware constrained neural substrates," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 1–8.

Shukla, R., Khoram, S., Jorgensen, E., Li, J., Lipasti, M., and Wright, S. (2018). Computing generalized matrix inverse on spiking neural substrate. *Front. Neurosci.* 12:115. doi: 10.3389/fnins.2018.00115

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv[Preprint].arXiv:1409.1556*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: the all convolutional net. *arXiv[Preprint].arXiv:1412.6806*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., et al. (2014). Going deeper with convolutions. *arXiv[Preprint].arXiv:1409.4842*.

Wen, W., Wu, C., Wang, Y., Nixon, K., Wu, Q., Barnell, M., et al. (2016). "A new learning method for inference accuracy, core occupation, and performance co-optimization on truenorth chip," in *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16 (New York, NY: ACM), 18:1–18:6.

Yazdani, R., Riera, M., Arnau, J., and González, A. (2018). "The dark side of DNN pruning," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)* (Los Angeles, CA), 790–801. doi: 10.1109/ISCA.2018.00071

Conflict of Interest Statement: ML has financial interest in Thalchemy corp. and is co-founder of the said corporation. Thalchemy corp. was not at all involved in this research project in any form.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 19 March 2019
doi: 10.3389/fnins.2019.00189

ReStoCNet: Residual Stochastic Binary Convolutional Spiking Neural Network for Memory-Efficient Neuromorphic Computing

Gopalakrishnan Srinivasan and Kaushik Roy*

Department of ECE, Purdue University, West Lafayette, IN, United States

In this work, we propose ReStoCNet, a residual stochastic multilayer convolutional Spiking Neural Network (SNN) composed of binary kernels, to reduce the synaptic memory footprint and enhance the computational efficiency of SNNs for complex pattern recognition tasks. ReStoCNet consists of an input layer followed by stacked convolutional layers for hierarchical input feature extraction, pooling layers for dimensionality reduction, and fully-connected layer for inference. In addition, we introduce residual connections between the stacked convolutional layers to improve the hierarchical feature learning capability of deep SNNs. We propose Spike Timing Dependent Plasticity (STDP) based probabilistic learning algorithm, referred to as Hybrid-STDP (HB-STDP), incorporating Hebbian and anti-Hebbian learning mechanisms, to train the binary kernels forming ReStoCNet in a layer-wise unsupervised manner. We demonstrate the efficacy of ReStoCNet and the presented HB-STDP based unsupervised training methodology on the MNIST and CIFAR-10 datasets. We show that residual connections enable the deeper convolutional layers to self-learn useful high-level input features and mitigate the accuracy loss observed in deep SNNs devoid of residual connections. The proposed ReStoCNet offers $>20\times$ kernel memory compression compared to full-precision (32-bit) SNN while yielding high enough classification accuracy on the chosen pattern recognition tasks.

Keywords: convolutional SNN, spiking ResNet, binary kernels, probabilistic STDP, unsupervised feature learning

OPEN ACCESS

Edited by:
Yansong Chua,
*Institute for Infocomm Research (A*STAR), Singapore*

Reviewed by:
Timothée Masquelier,
Centre National de la Recherche Scientifique (CNRS), France
Andrew Rowley,
University of Manchester, United Kingdom

****Correspondence:***
Gopalakrishnan Srinivasan
srinivg@purdue.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *29 November 2018*
Accepted: *18 February 2019*
Published: *19 March 2019*

Citation:
Srinivasan G and Roy K (2019) ReStoCNet: Residual Stochastic Binary Convolutional Spiking Neural Network for Memory-Efficient Neuromorphic Computing. Front. Neurosci. 13:189. doi: 10.3389/fnins.2019.00189

1. INTRODUCTION

The proliferation in real-time content generated by the ubiquitous battery-powered edge devices necessitates a paradigm shift in neural architectures to enable energy-efficient neuromorphic computing. Spiking Neural Networks (SNNs) offer a promising alternative toward realizing intelligent neuromorphic systems that require lower computational effort than the artificial neural networks. SNNs encode and communicate information in the form of sparse spiking events. The intrinsic sparse event-driven processing capability, which entails neuronal computations and synaptic weight updates only in the event of a spike fired by the constituting neurons, leads to improved energy efficiency in neuromorphic hardware implementations (Sengupta et al., 2019). Spike Timing Dependent Plasticity (STDP) (Bi and Poo, 1998) is a localized hardware-friendly plasticity mechanism used for unsupervised learning in SNNs. STDP-based learning rules (Song et al., 2000) modify the weight of a synapse interconnecting a pair of input (pre) and output (post) neurons depending on the degree of correlation between the respective spike times. The spike timing information is encoded in the bit-precision of the synaptic weight. In an effort to reduce the synaptic memory footprint, Suri et al. (2013), Querlioz et al. (2015), and Srinivasan et al. (2016) proposed two-layer fully-connected SNN composed of binary synaptic weights. The fully-connected SNN learns complete input representations rather than distinctive features making up the input patterns. As a result, it requires large number of trainable parameters to attain competitive classification accuracy (Diehl and Cook, 2015), which negatively impacts the scalability of such shallow SNNs for complex pattern recognition tasks.

We propose deep Residual Stochastic Binary Convolutional Spiking Neural Network, referred to as *ReStoCNet*, as a scalable architecture to achieve improved classification accuracy with compressed synaptic memory. ReStoCNet consists of an input layer followed by stacked convolutional layers with Leaky-Integrate-and-Fire (LIF) spiking non-linearity (Dayan and Abbott, 2001) for hierarchical input feature extraction, spatial pooling layers for dimensionality reduction, and one or more fully-connected layers for inference. We introduce residual or shortcut connections between the stacked convolutional layers, inspired by the organization of deep residual networks (He et al., 2016), in order to improve the representations learnt by the later convolutional layers. In addition, we enforce binary synaptic weights for the convolutional kernels during both training and inference. We propose STDP-based probabilistic learning rule, referred to as Hybrid-STDP (HB-STDP), incorporating Hebbian and anti-Hebbian learning mechanisms to train the binary kernels. Based on HB-STDP, a binary synaptic weight is probabilistically potentiated for small positive time difference between excitatory pre- and post-spikes, which is in agreement with the Hebbian learning theory (Hebb, 1949). On the other hand, it is probabilistically depressed for large positive time difference (anti-Hebbian in nature) or small negative time difference (Hebbian in nature) between the respective spikes. The spike timing information is essentially encoded in the synaptic switching probability, which is held constant within the Hebbian potentiation, Hebbian depression, and anti-Hebbian depression windows, and is zero elsewhere. We note that Suri et al. (2013) proposed an STDP-based learning rule employing constant switching probabilities, where the potentiation and depression windows extend over the entire STDP timing window. On the contrary, HB-STDP contains dead zone in the STDP timing window, where the switching probability is zero. We visually demonstrate the significance of dead zone for efficient feature learning using binary fully-connected SNN.

We present HB-STDP based layer-wise unsupervised training methodology for ReStoCNet, where we train the binary kernels interconnecting successive convolutional layers using HB-STDP. Once a given layer is trained, we forward propagate the spikes from the input through the trained layers and update the binary kernels of the following convolutional layer. After all the convolutional layers are trained, we feed the input dataset, estimate the spiking activations of the spatially pooled convolutional spike maps by accumulating the spikes at every time instant and decaying the resultant sum between successive spike timing instants, and pass them on to the fully-connected layer, trained using error backpropagation (Rumelhart et al., 1986), for inference. We validate the efficacy of ReStoCNet and the HB-STDP based unsupervised training methodology on the MNIST (LeCun et al., 1998) and CIFAR-10 datasets (Krizhevsky, 2009). We show that residual connections enable the deeper convolutional layers to extract useful high-level input features and effectively mitigate the accuracy degradation observed in deep SNNs devoid of residual connections (Lee et al., 2018b). We note that Masquelier and Thorpe (2007), Panda and Roy (2016), Lee et al. (2016), Stromatias et al. (2017), Srinivasan et al. (2018), Tavanaei et al. (2018), Kheradpisheh et al. (2018), Ferré et al. (2018), Thiele et al. (2018), Lee et al. (2018a,b), and Mozafari et al. (2018) have demonstrated convolutional SNNs composed of full-precision kernels. Recently, Sengupta et al. (2019) and Hu et al. (2018) presented residual SNNs, trained using error backpropagation with real-valued inputs and artificial ReLU neurons (Nair and Hinton, 2010), which are mapped to spiking neurons post training for energy-efficient inference. To the best of our knowledge, ReStoCNet is the first demonstration of STDP-trained deep residual convolutional SNN composed of binary kernels for complex pattern recognition tasks. We believe that ReStoCNet, with event-driven computing capability and memory-efficient learning with binary kernels trained using hardware-friendly probabilistic-STDP learning rule, offers a promising alternative for energy-efficient neuromorphic computing in battery-powered edge devices. Overall, the key contributions of our work are:

1. We propose ReStoCNet, a deep residual convolutional SNN composed of binary kernels, for memory-efficient neuromorphic computing.
2. We present HB-STDP, an STDP-based probabilistic learning rule incorporating Hebbian and anti-Hebbian learning mechanisms, for training the binary kernels constituting ReStoCNet in a layer-wise unsupervised manner for hierarchical input feature extraction.
3. We validate the efficacy of ReStoCNet on the MNIST and CIFAR-10 datasets, and show that residual connections enable the deeper convolutional layers to learn useful high-level input features and mitigate the accuracy loss incurred by STDP-trained deep SNNs without residual connections.

2. MATERIALS AND METHODS

2.1. ReStoCNet: Residual Stochastic Binary Convolutional Spiking Neural Network

ReStoCNet consists of an input layer followed by stacked convolutional layers for hierarchical input feature extraction, spatial pooling layers for dimensionality reduction, and one or more fully-connected layers for inference as illustrated in **Figure 1**. The pixels in the input image maps are converted to Poisson spike trains firing at a rate proportional to the corresponding pixel intensities. At any given time, the input spike maps are convolved with the binary kernels, which are constrained to logic states -1 (w_{low}) and $+1$ (w_{high}), to produce the convolutional output maps. The convolutional

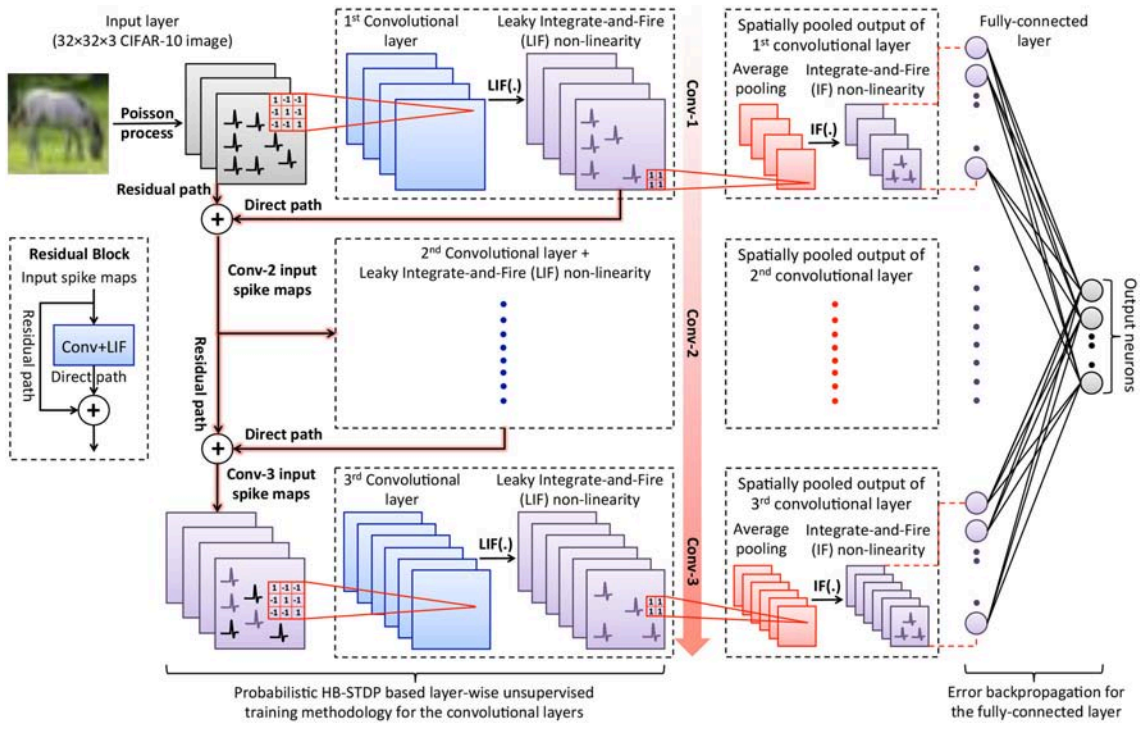

Input layer
(32×32×3 CIFAR-10 image)
Poisson process
Residual path
Direct path
1st Convolutional layer
Leaky Integrate-and-Fire (LIF) non-linearity
LIF(.)
Conv-1
Spatially pooled output of 1st convolutional layer
Average pooling
Integrate-and-Fire (IF) non-linearity
IF(.)
Fully-connected layer
Residual Block
Input spike maps
Conv+LIF
Direct path
Residual path
Conv-2 input spike maps
2nd Convolutional layer + Leaky Integrate-and-Fire (LIF) non-linearity
Conv-2
Spatially pooled output of 2nd convolutional layer
Output neurons
Conv-3 input spike maps
3rd Convolutional layer
Conv-3
Spatially pooled output of 3rd convolutional layer
Probabilistic HB-STDP based layer-wise unsupervised training methodology for the convolutional layers
Error backpropagation for the fully-connected layer

FIGURE 1 | Illustration of ReStoCNet consisting of an input layer followed by stacked convolutional layers with Leaky-Integrate-and-Fire (LIF) spiking non-linearity, which are interconnected via binary kernels. The deeper convolutional layers receive residual inputs that are summed up with direct inputs from the preceding convolutional layer as depicted in the inset. The binary kernels forming the convolutional layers are trained using probabilistic Hybrid-STDP (HB-STDP) based layer-wise unsupervised training methodology. After all the convolutional layers are trained, the respective spike maps are spatially pooled using average pooling with 2×2 unit-weight kernels followed by Integrate-and-Fire (IF) spiking non-linearity to produce the pooled spike maps. The spike trains of the pooling layers are low-pass filtered to obtain their spiking activations over the time period for which the input is presented, which are fed to the fully-connected layer, trained using error backpropagation, for inference.

outputs, referred to as post-synaptic currents, are fed to non-linear layer of Leaky-Integrate-and-Fire (LIF) spiking neurons (Dayan and Abbott, 2001). An LIF neuron integrates the post-synaptic current into its membrane potential, whose dynamics are described by

$$\tau_{mem}\frac{dV_{mem}}{dt} = -V_{mem} + I_{post} \tag{1}$$

where V_{mem} is the neuronal membrane potential, τ_{mem} is the membrane potential leak time constant, and I_{post} is the post-synaptic current. The LIF neuron emits a spike when its membrane potential exceeds a definite firing threshold after which the membrane potential is reset to zero. Every convolutional output map yields a corresponding spike map based on the LIF spiking neuronal dynamics, which is directly fed to the following convolutional layer. In addition, we introduce residual connections feeding into the deeper convolutional layers, which is inspired by the architecture of deep residual networks (He et al., 2016). The second convolutional layer receives residual connections from the input layer while the third convolutional layer receives residual connections from the input and first convolutional layer as shown in **Figure 1**. The residual connections feeding into a target convolutional layer perform identity mapping, i.e., the residual path spike maps are simply added to the direct path spike maps from the preceding convolutional layer and fed to the target convolutional layer. In the event of a mismatch in the number of spike maps (or channels) between the residual and direct paths, the spike maps in the residual path are replicated to be consistent with the number of channels in the direct path. Consider, for instance, the second convolutional layer that receives spike maps from the input layer via the residual path and the first convolutional layer via the direct path. Let us suppose that the input image pattern is stored in RGB colorspace. Consequently, each image pattern yields 3 input spike maps that needs to be summed up with the spike maps of the first convolutional layer, which typically contains more than 3 spike maps. Hence, the 3 input spike maps are replicated to match the number of spike maps in the first convolutional layer, summed up with the spike maps of the first convolutional layer, and fed to the second convolutional layer. Note that the summed spike maps from the residual and direct paths are constrained to unit magnitude to produce resultant spike maps feeding into the target convolutional layer. The binary kernels constituting the convolutional layers are trained using probabilistic Hybrid-STDP (HB-STDP) based layer-wise unsupervised training methodology. We find that the residual connections ensure rich and diverse inputs for deeper convolutional layers and enable them to self-learn useful high-level input features as shown in subsection 3.3. The improved feature learning capability mitigates the accuracy loss incurred by stacked convolutional layers without residual connections as experimentally validated in subsection 3.3 and enhances the scalability of deep SNNs.

After all the convolutional layers are trained, we feed the input dataset and spatially pool the spike maps of the convolutional layers. Spatial pooling is the mechanism used to suitably combine the neighboring pixels of a convolutional feature map to reduce the map size (height and width) while retaining the salient features. Spatial pooling also renders the network invariant to slight translations in the input features (Jaderberg et al., 2015). We perform a class of spatial pooling operation known as average pooling with 2×2 kernels composed of unit weights and stride length of 2 as detailed below. The spikes in every 2×2 non-overlapping region of the convolutional maps are summed up and normalized by the kernel size (4 for a 2×2 kernel) to produce the pooled output maps, which are then fed to a layer of Integrate-and-Fire (IF) spiking neurons to generate the pooled spike maps. An IF neuron integrates the input into its membrane potential and spikes if the membrane potential exceeds pre-specified threshold (θ_{pool}) after which the membrane potential is reset. The IF neurons, in effect, fire based on the average spiking activity of the spatially pooled convolutional spike maps. We low-pass filter the spike trains of the pooled maps by integrating the spikes at every time instant and decaying the resultant sum between successive spike timing instants to estimate their spiking activations over the time period for which the input is presented. The spiking activations of the pooled maps pertaining to all the convolutional layers are fed to the fully-connected layer composed of ReLU neurons (Nair and Hinton, 2010) for inference. This ensures that the input features learnt independently by the convolutional layers in an unsupervised manner are combined optimally by the fully-connected layer to yield the best accuracy. We note that LIF neurons can instead be used in the fully-connected layer, which can be trained using spike-based backpropagation algorithms (Lee et al., 2016, 2018a; Panda and Roy, 2016; Jin et al., 2018; Wu et al., 2018). In this work, we use fully-connected layer of ReLU neurons trained with backpropagation algorithm commonly used for deep learning networks since we are primarily interested in evaluating the efficacy of the proposed probabilistic HB-STDP based unsupervised training methodology for the convolutional layers that is detailed in the following subsection.

2.2. Hybrid-STDP (HB-STDP) for Binary Synaptic Weights

We propose STDP-based probabilistic learning rule, referred to as Hybrid-STDP (HB-STDP), integrating Hebbian and anti-Hebbian learning mechanisms to train the binary synaptic weights constituting an SNN. We present two versions of the HB-STDP learning rule, namely, excitatory HB-STDP (eHB-STDP) and inhibitory HB-STDP (iHB-STDP) to train the binary synaptic weights connecting excitatory and inhibitory pre-neurons, respectively, to excitatory post-neurons. An excitatory neuron is modeled as a neuron firing unit positive spikes while an inhibitory neuron fires unit negative spikes. Input image pixels with intensities ranging from 0 to 255 are mapped to excitatory pre-neurons firing unit positive spikes at a rate proportional to the respective pixel intensities. On the contrary, input images when pre-processed by normalizing the raw pixel intensities to zero mean and unit variance result in normalized images with positive and negative pixel intensities. The normalized pixels with negative intensities are mapped to inhibitory pre-neurons firing unit negative spikes. The normalized input maps containing excitatory and inhibitory pre-neurons offer richer spike-encoding of the image patterns, resulting in efficient STDP-based feature learning. We find that input normalization is critical for natural images like those from the CIFAR-10 dataset (Krizhevsky, 2009) that do not have clear separation between the region of interest and the background unlike digit patterns from the MNIST dataset (LeCun et al., 1998).

Binary synapses require a probabilistic learning rule to prevent rapid switching of the weights between the allowed levels, which could otherwise render the synapses memoryless. Both the proposed eHB-STDP and iHB-STDP learning rules map the time difference between a pair of pre- and post-spikes to the switching probability of the interconnecting binary synapse. We first detail the eHB-STDP learning rule for excitatory pre-neurons and subsequently discuss how the learning dynamics are adapted for inhibitory pre-neurons. According to eHB-STDP, if an excitatory pre-spike (at time instant, t_{pre}) triggers the post-neuron to fire (at time instant, t_{post}) and the difference between the respective spike times ($\Delta t = t_{post} - t_{pre}$) is smaller than a pre-specified time period (t_{Hebb_pot}), we switch the synapse from low to high ('L'→'H') state with a constant probability, p_{Hebb_pot}, as illustrated in **Figure 2A** and described by

$$P_{L\to H} = \begin{cases} p_{Hebb\_pot}, & \text{if } 0 < \Delta t \le t_{Hebb\_pot} \\ 0, & \text{for all other } \Delta t \end{cases} \tag{2}$$

where $P_{L\to H}$ is the probability of synaptic potentiation. Probabilistic synaptic potentiation is carried out for small time difference between causally related pre- and post-spikes following the Hebbian learning principle that can be summarized as "*neurons that fire together, must wire together*" (Lowel and Singer, 1992). Hence, the corresponding timing window is designated as the *Hebbian potentiation* window. On the other hand, probabilistic synaptic depression is carried out for large positive or small negative time difference between the pre- and post-spikes as specified by

$$P_{H\to L} = \begin{cases} p_{antiHebb\_dep}, & \text{if } \Delta t > 0 \cap \Delta t \ge t_{antiHebb\_dep} \\ p_{Hebb\_dep}, & \text{if } t_{Hebb\_dep} \le \Delta t \le 0 \\ 0, & \text{for all other } \Delta t \end{cases} \tag{3}$$

where $P_{H\to L}$ is the probability of synaptic depression. We depress the synapse from high to low state with a constant probability, $p_{antiHebb\_dep}$, if the time difference between causally related pre- and post-spikes is larger than $t_{antiHebb\_dep}$, which is anti-Hebbian in nature. Hence, the corresponding STDP

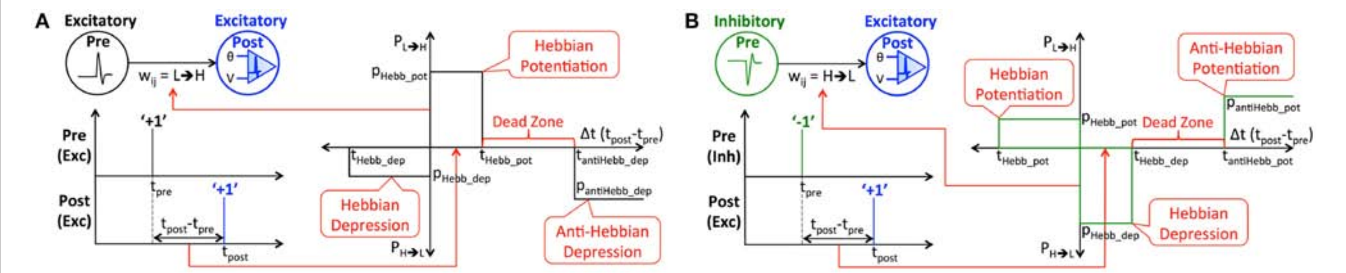


FIGURE 2 | (A) Illustration of eHB-STDP, an STDP-based probabilistic learning rule, incorporating Hebbian and anti-Hebbian learning mechanisms, for training the binary synaptic weights interconnecting excitatory pre- and post-neurons firing positive spikes. The synaptic weight is probabilistically potentiated for small positive time difference (Hebbian in nature) while it is probabilistically depressed for large positive (anti-Hebbian in nature) or small negative time difference (Hebbian in nature) between the pre- and post-spikes. The switching probability is held constant within the Hebbian potentiation, Hebbian depression, and anti-Hebbian depression windows, and is zero in the dead zone. **(B)** Illustration of iHB-STDP for binary synaptic weights connecting inhibitory pre-neurons firing negative spikes to excitatory post-neurons. The iHB-STDP dynamics are obtained by mirroring the eHB-STDP dynamics about the Δt ($t_{post} - t_{pre}$) axis.

timing window is referred to as the *anti-Hebbian depression* window. Anti-Hebbian depression enables the synapses to unlearn features lying outside the neuronal receptive field like noisy background in image patterns. Synaptic depression, in addition, is carried out with a probability, p_{Hebb_dep}, if a pre-spike follows a post-spike and the difference between the respective spike times lies within the negative *Hebbian depression* ($[t_{Hebb\_dep}, 0]$) window. It is important to note that eHB-STDP contains a dead zone in the STDP timing window, where the switching probability is zero, between the Hebbian potentiation and anti-Hebbian depression windows as depicted in **Figure 2A**. We find that expanding the anti-Hebbian depression window toward the Hebbian potentiation window leads to depression of moderately correlated features in addition to the weakly correlated ones. On the other hand, expanding the Hebbian potentiation window causes the synapses connecting a post-neuron to encode multiple overlapping input features, which negatively impacts the selectivity of the post-neuron and degrades the inference capability of the SNN. The dead zone, in effect, ensures that binary synapses learn and retain strongly correlated input features and unlearn only the weakly correlated ones by facilitating optimal balance between the potentiation and depression updates. We visually demonstrate the significance of dead zone for efficient feature learning using binary fully-connected SNN in subsection 3.1.

Next, we discuss how the eHB-STDP dynamics are adapted for binary synapses connecting inhibitory pre-neurons firing negative spikes. The iHB-STDP dynamics (shown in **Figure 2B**) are obtained by symmetrically inverting the eHB-STDP dynamics (shown in **Figure 2A**) about the Δt ($t_{post} - t_{pre}$) axis. As a result, the erstwhile potentiation windows are converted to depression windows, and vice versa. According to iHB-STDP, if an inhibitory pre-spike causes the post-neuron to fire and the spike timing difference is smaller than a pre-specified time period, we probabilistically depress the binary synaptic weight. This ensures that the strongly correlated inhibitory (negative) pre-spike modulated by the depressed synaptic weight causes an effective increase in the post-neuronal membrane potential, thereby improving the chances of a post-spike at subsequent time instants. Probabilistic synaptic depression enables a post-neuron to integrate the small positive time difference between an inhibitory pre-spike and the ensuing post-spike, which conforms to the Hebbian learning theory. Probabilistic synaptic potentiation, on the other hand, causes an inhibitory pre-spike modulated by the synaptic weight to lower the post-neuronal membrane potential, thus reducing the chances of a post-spike at subsequent time instants. Hence, it is carried out for large positive time difference (anti-Hebbian in nature) or small negative time difference (Hebbian in nature) between the pre- and post-spikes. The iHB-STDP learning rule for inhibitory pre-neurons effectively incorporates the learning dynamics of eHB-STDP for excitatory pre-neurons by mirroring the potentiation and depression windows about the Δt axis.

In this work, we use trace-based technique to estimate spike timing differences as it is commonly adopted for efficient implementation of STDP learning rules (Diehl and Cook, 2015). For instance, the positive time difference between a pair of pre- and post-spikes is estimated by generating an exponentially decaying pre-trace (with time constant τ_{pre}) that is reset to unity at the time instant of a pre-spike, and sampling it in the event of a post-spike. Smaller the time difference between the pre- and post-spikes, larger is the sampled pre-trace, and vice versa. Every pre-neuron has a pre-trace that is sampled upon a post-spike to obtain the positive spike timing difference. Likewise, every post-neuron has a post-trace (with time constant τ_{post}) that is sampled upon a pre-spike to obtain the negative spike timing difference. As a result, the eHB-STDP (iHB-STDP) hyperparameters, namely, t_{Hebb_pot} (t_{Hebb_dep}), $t_{antiHebb\_dep}$ ($t_{antiHebb\_pot}$), and t_{Hebb_dep} (t_{Hebb_pot}) are mapped to pre_{Hebb_pot} (pre_{Hebb_dep}), $pre_{antiHebb\_dep}$ ($pre_{antiHebb\_pot}$), and $post_{Hebb\_dep}$ ($post_{Hebb\_pot}$), respectively.

2.3. Unsupervised Training Methodology for the Convolutional Layers

We train the binary kernels forming ReStoCNet in a layer-wise unsupervised manner using the proposed probabilistic e/iHB-STDP learning rule. Consider a $k \times k$ binary kernel ($kernel_{ij}^{l}$) connecting the i^{th} input spike map in layer "$l-1$" (map_{i}^{l-1})

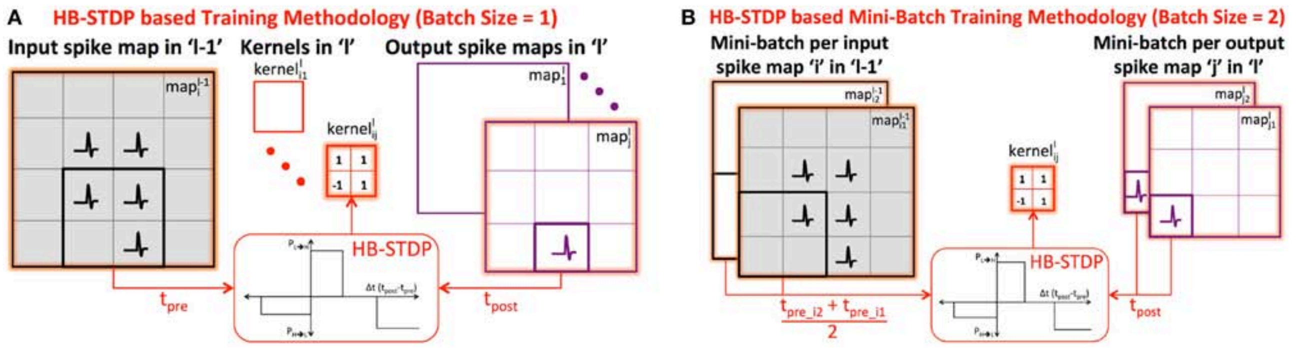

A HB-STDP based Training Methodology (Batch Size = 1)
Input spike map in 'l-1'
Kernels in 'l'
Output spike maps in 'l'
B HB-STDP based Mini-Batch Training Methodology (Batch Size = 2)
Mini-batch per input spike map 'i' in 'l-1'
Mini-batch per output spike map 'j' in 'l'
HB-STDP
t_pre
t_post
(t_pre_i2 + t_pre_i1)/2

FIGURE 3 | (A) Illustration of HB-STDP based unsupervised training methodology for $kernel_{ij}^{l}$ connecting the i^{th} input spike map in layer "$l-1$" (map_i^{l-1}) to the j^{th} output spike map in layer "l" (map_j^l). The $kernel_{ij}^{l}$ is updated using HB-STDP based on the spike timing difference between the spiking post-neuron in output map_j^l and the corresponding pre-neurons in input map_i^{l-1}. The HB-STDP based weight updates are carried out on all the kernels in layer "l" based on the respective input and output maps. **(B)** Illustration of HB-STDP based mini-batch training methodology for mini-batch size of 2. The $kernel_{ij}^{l}$ is now updated based on the average spike timing difference between the spiking post-neurons in the output mini-batch (map_{j1}^l and map_{j2}^l) and the respective pre-neurons in the input mini-batch (map_{i1}^{l-1} and map_{i2}^{l-1}).

to the j^{th} output spike map in layer "l" (map_j^l) as shown in **Figure 3A**. Let us suppose that a post-neuron in the output map_j^l spikes at a particular time instant: the kernel weights are then probabilistically updated based on the time difference between the post-spike and the corresponding $k \times k$ pre-spikes in the input map_i^{l-1}. We use the eHB-STDP learning rule for excitatory pre-neurons and iHB-STDP learning rule for inhibitory pre-neurons as described in subsection 2.2. If multiple post-neurons in the output map_j^l spike, we update $kernel_{ij}^l$ based on the average spike timing difference between the spiking post-neurons and the respective pre-neurons, which leads to generalized feature learning. However, in order to achieve optimal generalization performance, we average the spike timing differences computed with fixed stride, known as $STDP_{stride}$, over the output map_j^l. As an example, for $STDP_{stride}$ of 2, we average the spike timing differences computed between every alternate spiking post-neuron in output map_j^l and the respective pre-neurons. Larger the $STDP_{stride}$, fewer is the number of post-neurons whose spike timing difference estimates are averaged to update the kernel. Consequently, there is loss of generality and added specificity in the features learnt by the kernel for larger $STDP_{stride}$. We experimentally determine the $STDP_{stride}$ for optimal generalization performance that yields the highest test accuracy for a given pattern recognition task.

STDP-based learning is typically performed in an online manner by feeding the input patterns sequentially. STDP-based online learning has been shown to work well particularly for two-layer fully-connected SNNs, where each output or excitatory neuron learns to spike exclusively for a unique class of input patterns by encoding a general input representation in the input to excitatory synaptic weights (Diehl and Cook, 2015). Convolutional SNNs, on the other hand, require each kernel to extract features shared across different input classes. In order to enable the kernel to extract general features characterizing different input classes, we perform mini-batch learning following recent works by Lee et al. (2018b) and Ferré et al. (2018). The proposed HB-STDP based mini-batch training methodology is illustrated in **Figure 3B**, where the $kernel_{ij}^l$ is now shared by a mini-batch of i^{th} input map in layer "$l-1$" (input mini-batch) and j^{th} output map in layer "l" (output mini-batch). We first average the spike timing differences between the spiking post-neurons and the respective pre-neurons, estimated using fixed $STDP_{stride}$, over each output map in the mini-batch to obtain the resultant spike timing difference per output map in the mini-batch. We subsequently average the resultant spike timing differences of the output maps across the mini-batch and probabilistically update $kernel_{ij}^l$ using HB-STDP as shown in **Figure 3B** for a specific post-neuron in the output mini-batch. At every time instant, the HB-STDP driven mini-batch weight updates are carried out on all the kernels in a given layer. This process is repeated over the entire time duration, T_{STDP}, for which the training patterns are presented.

Finally, in order to ensure that different kernels in a layer learn diverse input features, we incorporate the uniform firing threshold adaptation scheme proposed by Lee et al. (2018b) and dropout (Srivastava et al., 2014) for the output maps. In the beginning of training, the firing threshold of all the post-neurons in every output mini-batch is reset to zero. When a mini-batch of training patterns is presented, multiple post-neurons in an output mini-batch spike and encode definite input features in the kernel weights. We then increase the firing threshold of all the post-neurons in the output mini-batch by an amount $\Delta thresh$, which is specified by

$$\Delta thresh = \beta_{thresh} \times \frac{output\ spike\ count}{output\ map\ size} \tag{4}$$

where β_{thresh} is the rate of threshold increase, *output spike count* is the number of spikes per output map summed over the mini-batch, and *output map size* is the product of the height and width of the output maps. The amount of threshold increase depends on the *output spike count* normalized by the *output map size* to account for the drop in spiking activity of the output maps across successive convolutional layers due to gradual reduction in the respective sizes. Higher the normalized spiking activity of the output mini-batch, greater is the corresponding increase in its firing threshold, and vice versa. Firing threshold adaptation effectively regulates the spiking activity of the output mini-batch and provides an opportunity for the hitherto dormant output mini-batches to spike and learn, thereby ensuring that no single output mini-batch completely dominates the learning process during a mini-batch training iteration. In addition, we introduce dropout (Srivastava et al., 2014) for the output maps to achieve diversity in feature learning across successive mini-batch training iterations. At the beginning of every training iteration, we randomly drop a fraction of output mini-batches based on the dropout probability, p_{drop}, by forcing the respective spike outputs to zero. Dropout ensures that the same output mini-batch does not spike repeatedly for every training iteration, thereby promoting diversity in feature learning among the kernels in a layer. Once a layer is trained, we propagate the spikes from the input through the trained layers, and update the kernels and firing thresholds of the output maps in the following layer using the presented training methodology. The training process is repeated for all the convolutional layers in ReStoCNet.

2.4. Supervised Training Methodology for the Fully-Connected Layer

After all the convolutional layers are trained, we pool the respective spike maps using average pooling as detailed in subsection 2.1. We then low-pass filter the spike trains of the pooled maps, by integrating the spike outputs at every time instant and decaying the resultant sum between successive time instants, to obtain their spiking activations as described in Lee et al. (2016, 2018a) and specified by

$$\begin{aligned} pool^l_{lpf}(t) &= e^{-\frac{\Delta t_{sim}}{\tau_{lpf}}} \times pool^l_{lpf}(t - \Delta t_{sim}) + pool^l(t) \\ pool^l_{out} &= \frac{pool^l_{lpf}(T_{sim})}{T_{sim}} \end{aligned} \tag{5}$$

where $pool^l_{lpf}(t)$ is the low-pass filtered output of the pooled spike map $pool^l(t)$ in layer "l" at any given time t, τ_{lpf} is the low-pass filter time constant, Δt_{sim} is the simulation time-step, T_{sim} is the simulation period for which the input patterns are presented, and $pool^l_{out}$ is the spiking activation of the pooled map in layer "l" over the simulation period. The spiking activation thus obtained accounts for the highly non-linear leaky-integrate-and-fire and membrane potential reset dynamics of the spiking neurons in the convolutional layers. The spiking activations of the pooled maps of all the convolutional layers are concatenated and fed to the fully-connected layer, trained using error backpropagation (Rumelhart et al., 1986), for inference. We use full-precision synaptic weights in the fully-connected layer to comprehensively validate the efficacy of the proposed probabilistic HB-STDP learning rule for training the binary kernels in the convolutional layers. The full-precision synaptic weights can be binarized using algorithms proposed for training binary deep learning networks (Courbariaux et al., 2015; Rastegari et al., 2016; Hubara et al., 2017). It is important to note that the presented HB-STDP based learning methodology effectuates plasticity by probabilistically switching the binary weights, thereby precluding the need to store the full-precision weights during training. Binarization algorithms for deep learning networks, on the other hand, update the full-precision weights during training, which are subsequently binarized for forward propagation and computing the error gradients.

3. RESULTS

We first validate the efficacy of HB-STDP, by visually demonstrating the significance of having distinct potentiation and depression windows separated by a dead zone for efficient feature learning, using two-layer binary fully-connected SNN trained on the MNIST dataset. We then comprehensively evaluate ReStoCNet and the presented HB-STDP based unsupervised mini-batch training methodology on the MNIST and CIFAR-10 datasets. We show that the residual connections are critical to achieving efficient unsupervised learning in deeper convolutional layers and minimizing the accuracy degradation incurred by STDP-trained deep SNNs without residual connections. We use the classification accuracy on the test set and the synaptic memory compression obtained by using binary kernels as the evaluation metrics for ReStoCNet compared to full-precision (32-bit) SNN under iso-accuracy conditions.

3.1. Two-Layer Binary Fully-Connected SNN for MNIST Digit Recognition

The binary fully-connected SNN (Diehl and Cook, 2015) consists of an input layer fully-connected via binary synapses to neurons in the excitatory layer, which are connected in a one-to-one manner to neurons in the subsequent inhibitory layer. Each inhibitory neuron laterally inhibits all the excitatory neurons except the one from which it receives a forward connection. Lateral inhibition facilitates competitive learning and enables each excitatory neuron to spike exclusively and recognize a unique class of input patterns. The input to excitatory synaptic weights are trained using three different configurations of the eHB-STDP learning rule that are enumerated below:

1. eHB-STDP – This is the proposed eHB-STDP learning rule containing distinct Hebbian potentiation and anti-Hebbian depression windows separated by a dead zone as shown in **Figure 4A**.
2. eHB-STDP2 – This is a variant of the eHB-STDP learning rule where the dead zone is replaced with a wider Hebbian potentiation window as depicted in **Figure 4B**.
3. eHB-STDP3 – This is an alternative variant of the eHB-STDP rule where the dead zone is replaced with a wider anti-Hebbian depression window as illustrated in **Figure 4C**.

Note that the excitatory↔inhibitory synaptic weights are fixed *a priori* and are not subjected to STDP-based learning. We simulated the fully-connected SNN using BRIAN (Goodman and Brette, 2008), which is an open-source SNN simulation framework, on the MNIST dataset. The input image pixels are converted to Poisson spike trains firing at a rate constrained between 0 and 63.75 Hz depending on

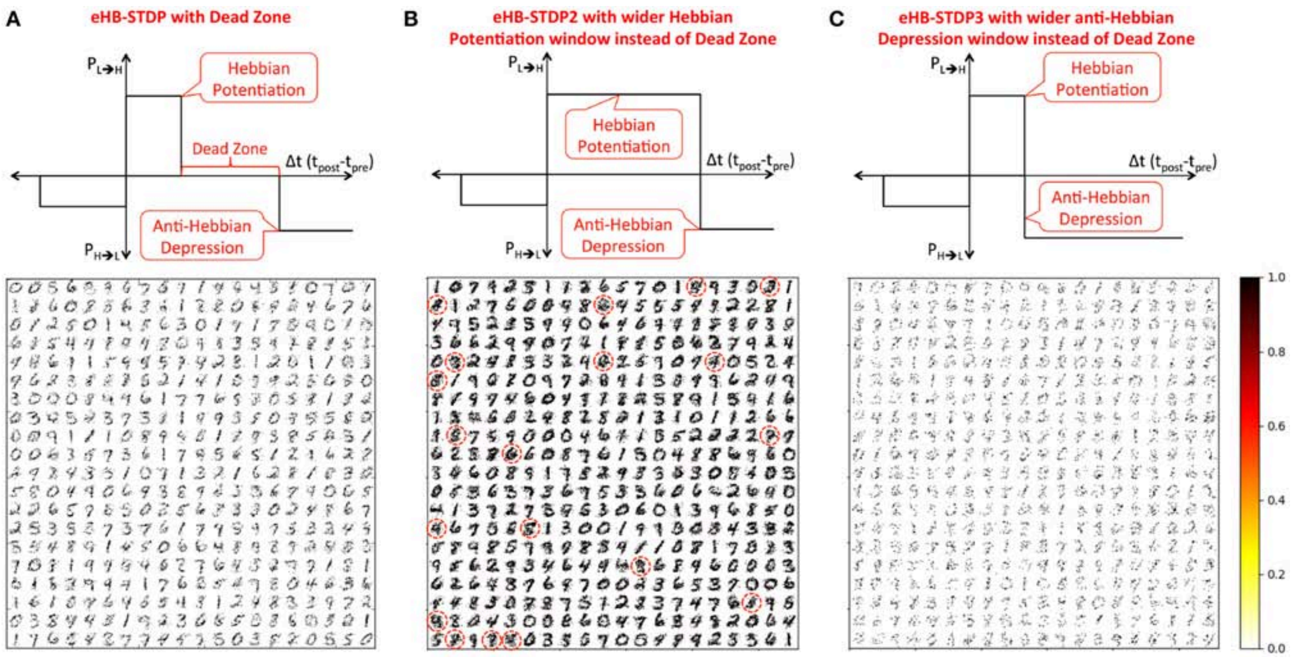


FIGURE 4 | MNIST digit representations (re-arranged in 28×28 format) learnt by the synapses connecting the input to each excitatory neuron in a binary fully-connected SNN of 400 neurons (arranged in 20×20 grid). The binary fully-connected SNN is trained using **(A)** the proposed eHB-STDP containing distinct Hebbian potentiation and anti-Hebbian depression windows separated by a dead zone, **(B)** eHB-STDP2 where the dead zone is replaced with a wider Hebbian potentiation window, and **(C)** eHB-STDP3 where the dead zone is replaced with a wider anti-Hebbian depression window.

the respective pixel intensities for a simulation period of 350 ms. Note that the simulation time-step is 0.5 ms. We use the spiking neuronal model detailed in Diehl and Cook (2015) whose parameters are adopted from Jug (2012). The eHB-STDP hyperparameters used in our simulations are listed in **Table 1**.

We first train a binary fully-connected SNN of 400 excitatory neurons using the three different eHB-STDP configurations on 3500 MNIST digit patterns. **Figure 4A** illustrates that eHB-STDP causes each excitatory neuron to self-learn general representation of a unique digit in the input to excitatory synaptic weights. On the other hand, eHB-STDP2, with a wider Hebbian potentiation window instead of the dead zone, causes certain excitatory neurons to self-learn overlapping input representations as highlighted in **Figure 4B**. Overlapping input representations negatively impact the selective spiking behavior of the excitatory neurons for specific input classes and degrade the recognition capability of the SNN. The final eHB-STDP configuration, eHB-STDP3, leads to insufficient representation learning as depicted in **Figure 4C** due to the dominance of synaptic depression over synaptic potentiation weight updates. Thus, the proposed eHB-STDP learning rule offers superior representation learning capability compared to the explored variants by maintaining optimal balance between the potentiation and depression weight updates. This is further corroborated by the accuracy results shown in **Figure 5A**, which is evaluated as explained below. At the end of eHB-STDP based training, each excitatory neuron is tagged as having learnt the class of input patterns for which it spiked the most during the training phase. A test pattern is predicted to belong to the class (or tag) represented by the group of neurons with the highest average spike count over the simulation period. The binary fully-connected SNN of 400 neurons trained using eHB-STDP yielded 79.94% accuracy on the MNIST test set, which is higher by >8% compared to that achieved using the remaining eHB-STDP variants. The accuracy can be further improved by increasing the number of excitatory neurons as shown in **Figure 5B**. We now estimate the *synaptic memory compression* offered by the binary SNN compared to full-precision (32-bit) SNN, which is specified by

$$\textit{synaptic memory compression} = \frac{\#\textit{input neurons} \times \#\textit{excitatory neurons}_{\textit{full-precisionSNN}} \times 32}{\#\textit{input neurons} \times \#\textit{excitatory neurons}_{\textit{binarySNN}} \times 1} \quad (6)$$

where *#input neurons* is 784 for the MNIST dataset. **Figure 5B** indicates that binary SNN of 6400 neurons offers comparable accuracy (∼92%) to that provided by full-precision (32-bit) SNN of 1600 neurons (Diehl and Cook, 2015), leading to 8× synaptic memory compression under iso-accuracy conditions. Note that the accuracy of ∼92% is higher than that reported in related works for binary fully-connected SNN, trained using probabilistic STDP-based learning rules, as shown in **Table 2**.

However, the fully-connected SNN introduces scalability issues as the network depth is increased due to explosion in the number of trainable parameters. We demonstrate ReStoCNet, which is a scalable multilayer convolutional SNN composed of binary kernels, trained using the optimal e/iHB-STDP based unsupervised mini-batch training methodology.

3.2. ReStoCNet for MNIST Digit Recognition

The MNIST dataset contains 60,000 training patterns and 10,000 test patterns of handwritten digits that are stored as 28×28 Grayscale images. In this work, we developed a custom simulation framework using Pytorch (Paszke et al., 2017) to evaluate ReStoCNet and the presented HB-STDP based unsupervised training methodology. The simulation parameters for the Leaky-Integrate-and-Fire (LIF) neuron in the convolutional layers and the Integrate-and-Fire (IF) neuron in the spatial pooling layers are shown in **Table 3**. The binary kernels in every convolutional layer are initialized to logic high state (w_{high}) with a probability, p_{high}, which is specified by

$$p_{high} = \sqrt{\frac{\alpha_{weight\_init}}{fan\_in + fan\_out}} \tag{7}$$

where α_{weight_init} is the proportionality constant controlling p_{high}, and fan_in and fan_out are the total number of input and output synaptic weights, respectively, for a given convolutional layer. The remaining kernel weights in the convolutional layer are initialized to logic low state (w_{low}). The firing threshold of the LIF neurons in every convolutional layer are initialized to zero.

We first simulated a 16C3-2P-10FC ReStoCNet, composed of single convolutional layer with 16 maps and 3×3 binary kernels followed by pooling layer whose spiking activations are directly fed to the final softmax layer. The input image pixels are mapped to excitatory pre-neurons firing at a rate constrained between 0 and 200 Hz depending on the corresponding pixel intensities. The eHB-STDP model parameters are provided in **Table 3**. We trained the convolutional layer in ReStoCNet using 2,000 MNIST digit patterns with a mini-batch size of 200. We thereafter fed the entire training dataset to ReStoCNet, spatially pooled the spike maps of the convolutional layer, and low-pass filtered the pooled spike trains over a simulation period of 100 ms to estimate their spiking activations. The pooling layer spiking activations are passed on to the fully-connected softmax layer, which is trained using the Adam optimizer (Kingma and Ba, 2014) and cross-entropy loss function for 100 epochs. The training parameters used for the fully-connected layer are mentioned in **Table 4**. The shallow ReStoCNet yielded an accuracy of 95.21% on the MNIST test set, which increased to 98.22% for a wider 36C3-2P-10FC ReStoCNet in which the convolutional layer is trained using 10,000 MNIST digit patterns. Further improvement in accuracy is obtained by augmenting the classifier in ReStoCNet with an additional fully-connected layer of 128 neurons prior to the softmax output layer as shown in **Figure 6**, which indicates that 36C3-2P-128FC-10FC ReStoCNet offers an improved accuracy

TABLE 1 | Simulation parameters for training the binary fully-connected SNN on the MNIST dataset.

Parameters	Values
Simulation time-step, Δt_{sim}	0.5 ms
Simulation period, T_{sim}	350 ms
Maximum input spike rate	63.75 Hz
Pre-trace time constant, τ_{pre}	20 ms
Post-trace time constant, τ_{post}	20 ms
pre_{Hebb_pot} (eHB-STDP)	0.85
$pre_{antiHebb\_dep}$ (eHB-STDP)	0.10
$post_{Hebb\_dep}$ (eHB-STDP)	0.80
p_{Hebb_pot} (eHB-STDP)	0.08
$p_{antiHebb\_dep}$ (eHB-STDP)	0.06
p_{Hebb_dep} (eHB-STDP)	0.005
Maximum synaptic weight (w_{high})	1.0
Minimum synaptic weight (w_{low})	0.0

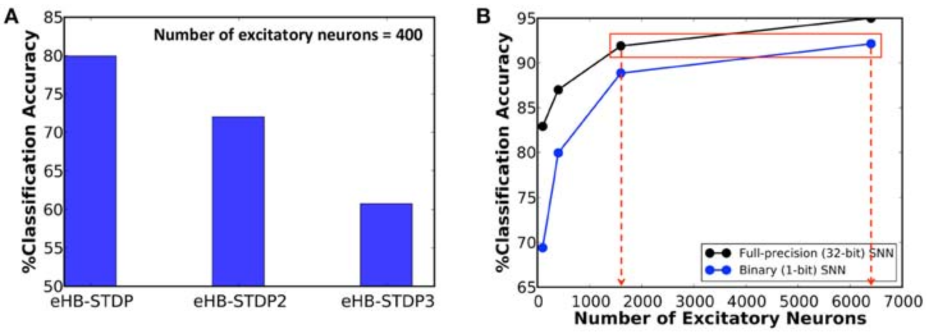


FIGURE 5 | (A) Classification accuracy of binary fully-connected SNN of 400 excitatory neurons trained using the three different eHB-STDP configurations illustrated in **Figure 4**. **(B)** Classification accuracy of binary fully-connected SNN, trained using the proposed eHB-STDP learning rule, compared to full-precision (32-bit) SNN (Diehl and Cook, 2015) for different network sizes.

TABLE 2 | Classification accuracy of binary fully-connected SNNs on the MNIST test set.

Model	#Excitatory neurons	Training methodology	Accuracy (%)
Binary SNN (Querlioz et al., 2015)	50	Probabilistic Rectangular STDP	60
Binary SNN (Srinivasan et al., 2016)	400	Probabilistic Exponential STDP	70.15
Binary SNN (our work)	400	Probabilistic eHB-STDP	79.94
Binary SNN (our work)	6400	Probabilistic eHB-STDP	92.14

TABLE 3 | Simulation parameters for training the convolutional layers in ReStoCNet.

Parameters	Values		
	C1	C1	C2/C3
Input dataset	MNIST	CIFAR-10	CIFAR-10
Maximum synaptic weight (w_{high})	+1.0	+1.0	+1.0
Minimum synaptic weight (w_{low})	−1.0	−1.0	−1.0
Weight initialization constant (α_{weight_init})	75	30	30
Simulation time-step, Δt_{sim}	1 ms	1 ms	1 ms
Simulation period for STDP, T_{STDP}	25 ms	25 ms	25 ms
Maximum input spike rate for STDP	200 Hz	200 Hz	500 Hz
Dropout probability for STDP, p_{drop}	0.5	0.5	0.5
$STDP_{stride}$	5	5	5
Pre-trace decay time constant, τ_{pre}	1.45 ms	1.45 ms	1.45 ms
pre_{Hebb_pot} (eHB-STDP)	0.50e-1	0.20e-1	0.20e-1
$pre_{antiHebb\_dep}$ (eHB-STDP)	0.50e-2	0.50e-2	0.50e-2
p_{Hebb_pot} (eHB-STDP)	0.01	0.05	0.05/25
$p_{antiHebb\_dep}$ (eHB-STDP)	0.01	0.01	0.01/25
p_{Hebb_dep} (eHB-STDP)	0	0	0
pre_{Hebb_dep} (iHB-STDP)	–	0.20e-1	0.20e-1
$pre_{antiHebb\_pot}$ (iHB-STDP)	–	0.50e-2	0.50e-2
p_{Hebb_dep} (iHB-STDP)	–	0.05	0.05/25
$p_{antiHebb\_pot}$ (iHB-STDP)	–	0.01	0.01/25
p_{Hebb_pot} (iHB-STDP)	–	0	0
Leaky-Integrate-and-Fire (LIF) neuron leak time constant, τ_{mem}	9.5 ms	9.5 ms	9.5 ms
Rate of increase of LIF neuronal firing threshold, β_{thresh}	6e-4	6e-4	6e-4 (C2) 8e-4 (C3)
Integrate-and-Fire (IF) neuron pooling threshold, θ_{pool}	0.80	0.80	0.80
Simulation period to estimate spiking activation, T_{sim}	100 ms	100 ms	100 ms
Maximum input spike rate to estimate spiking activation	500*Hz*	500*Hz*	500*Hz*
Low-pass filter time constant to estimate spiking activation, τ_{lpf}	99.5 ms	99.5 ms	99.5 ms

TABLE 4 | Simulation parameters for training the fully-connected layer in ReStoCNet.

Parameters	Values	
	MNIST	CIFAR-10
Batch size	256	256
Number of epochs	100	100
Learning rate (Adam)	1.5e-3	1.0e-4
betas (Adam)	(0.9, 0.999)	(0.9, 0.999)
eps (Adam)	1e-8	1e-8
Weight decay (Adam)	0	0
Dropout probability	0.5	0.5

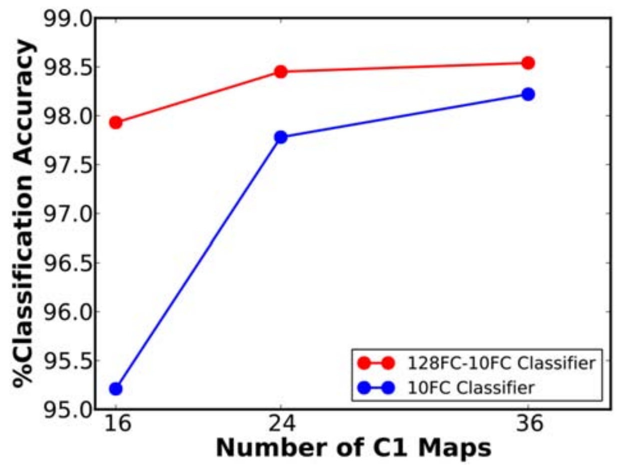


FIGURE 6 | Classification accuracy of ReStoCNet, composed of single convolutional layer followed by a pooling layer and one or more fully-connected layers, vs. the number of output (C1) maps, on the MNIST test set.

of 98.54% on the MNIST test set. Note that we did not simulate deep ReStoCNets for MNIST digit recognition since the shallow networks yield >98% accuracy, and that any further increase in the depth of STDP-trained convolutional layers would not provide commensurate improvements in the classification accuracy.

3.3. ReStoCNet for CIFAR-10 Image Recognition

The CIFAR-10 dataset contains 50,000 training images and 10,000 test images, 32×32×3 in dimension, spanning 10 output classes. We pre-processed the CIFAR-10 images using global contrast normalization followed by ZCA whitening (Krizhevsky, 2009). Global contrast normalization is performed by subtracting and scaling the pixel intensities of each input channel by the corresponding mean and standard deviation computed over the training set. The normalized image is then transformed by multiplying with whitening filters as explained in Krizhevsky (2009), which enables a network to learn higher-order pixel correlations. **Figure 7** illustrates a few original and pre-processed images from the CIFAR-10 dataset. The simulation parameters used for training the convolutional layers are provided in **Table 3** while those used for training the fully-connected layer are listed in **Table 4**. The binary kernels and firing thresholds of the convolutional layers are initialized as described in subsection 3.2.

In our first experiment, we simulated a 36C3-2P-1024FC-10FC ReStoCNet, designated as ReStoCNet-1, consisting of a single convolutional layer with 36 maps and 3×3 binary kernels followed by fully-connected layer containing 1024 ReLU neurons and a final softmax layer with 10 output neurons. The pre-processed CIFAR-10 images are composed of pixels with positive and negative intensities, which are, respectively, mapped to excitatory and inhibitory pre-neurons firing at a rate constrained between 0 and 200 Hz depending on the absolute value of the corresponding pixel intensities. The e/iHB-STDP model parameters are listed in **Table 3**. Note that the e/iHB-STDP switching probability is set to zero in the negative STDP timing window to facilitate optimal balance between the potentiation and depression updates for a smaller 3×3 kernel shared by 32×32 pre-neurons in the input map and 30×30 post-neurons in the convolutional map. The binary kernels in ReStoCNet-1 are trained using 5,000 images, with mini-batch size of 200, for simulation period of 25 ms per mini-batch training iteration. Note that we used a simulation time-step of 1 ms. **Figure 8A** illustrates the low-level input features self-learnt by the binary kernels, enabled by the e/iHB-STDP based unsupervised training methodology. The shallow ReStoCNet-1, wherein the fully-connected layer is trained on the entire dataset, yielded 64.31% test accuracy that is higher than an accuracy of 59.42% obtained using randomly initialized binary kernels and zero firing thresholds in the convolutional layer. In order to determine if accuracy loss is incurred as a result of using binary kernels, we trained ReStoCNet-1 composed of full-precision (32-bit) kernels using standard exponential STDP rule (Song et al., 2000) with learning rate of 0.01 for the positive STDP timing window and 0 for the negative STDP timing window. ReStoCNet-1 with full-precision kernels provided 64.30% test accuracy, which is comparable to that obtained using binary kernels. **Figure 8B** shows that the test accuracy improves with the number of maps in the convolutional layer. As explained in subsection 2.3, the classification accuracy of ReStoCNet has a strong dependence on the chosen $STDP_{stride}$ used for computing the average spike timing difference of the spiking post-neurons in the convolutional maps. **Figure 8C** indicates that the accuracy of ReStoCNet-1 degrades for $STDP_{stride}$ smaller than 4 or greater than 5. If the $STDP_{stride}$ is small, the binary kernels are updated based on the spike timing difference averaged over large number of spiking post-neurons in the convolutional maps, leading to degradation in the learnt features. On the contrary, if the $STDP_{stride}$ is large, the binary kernels are updated based on the spike timing difference estimates of few post-neurons, leading to loss of generality in the learnt features. We use the optimal $STDP_{stride}$ of 5 for all the ReStoCNet experiments presented in this work.

Next, we simulated a 36C3-36C3-2P-1024FC-10FC ReStoCNet, designated as ReStoCNet-2, composed of two convolutional layers, each with 36 maps and 3×3 binary kernels. The first convolutional layer is trained as described in the previous paragraph. The binary kernels and firing thresholds of the second convolutional layer are trained using a different subset of 5,000 CIFAR-10 images with a mini-batch size of 200. Note that the e/iHB-STDP hyperparameters are similar for both the convolutional layers except the synaptic switching probabilities, which are scaled down for the second convolutional layer as shown in **Table 3**. The lower switching probabilities for the second convolutional layer accounts for the fact that every constituting post-neuron receives weighted input from 36 maps each in the residual and direct paths while a post-neuron in the first convolutional layer receives weighted input from just the 3 maps in the input layer. We simulated two versions of ReStoCNet-2: one without residual connections and the other with residual connections from the input to second convolutional layer. **Figure 9** shows that ReStoCNet-2 with residual connections learns diverse high-level input features compared to the one without residual connections. As a result, ReStoCNet-2 with residual connections yielded 65.79% accuracy, which is roughly 1.5% higher than that provided by ReStoCNet-2 without residual connections as well as ReStoCNet-1. This begs the following question: *is ReStoCNet-2 yielding higher accuracy that ReStoCNet-1 just due to increased number of synaptic weights in the fully-connected layer as a consequence of concatenating the pooled spiking activations of both the convolutional layers?* To answer this question, we compare ReStoCNet-2, in which the spiking activations of the 72 pooled maps are fed to a fully-connected layer of 1024 neurons, with ReStoCNet-1 in which the spiking activations of the 36 pooled maps are fed to a larger fully-connected layer of 2048 neurons. **Figure 9C** indicates that ReStoCNet-2 offers higher accuracy than that provided by ReStoCNet-1 with 2048 neurons in the fully-connected layer, which is a testament to the improved feature learning capability of the second convolutional layer in the presence of residual inputs. **Figure 9D** shows that ReStoCNet-2 provides only modest improvement in accuracy as the number of output maps is increased in the second convolutional layer. The accuracy limitation is caused by the inability of the unsupervised training methodology to effectively optimize an over-parameterized network.

Finally, we evaluated a deeper 36C3-36C3-36C3-2P-1024FC-10FC ReStoCNet, referred to as ReStoCNet-3, composed of three convolutional layers as depicted in **Figure 1**. We inverted the residual inputs to the third convolutional layer to ensure diversity in the residual maps received by the second and third layers from the input layer. We trained the third convolutional layer with the same hyperparameters (shown in **Table 3**) as those used for training the second convolutional layer, albeit on a different subset of 5,000 images from the CIFAR-10 dataset. In addition to ReStoCNet-3 (with residual connections), wherein the pooled spiking activations of all the convolutional layers are used for

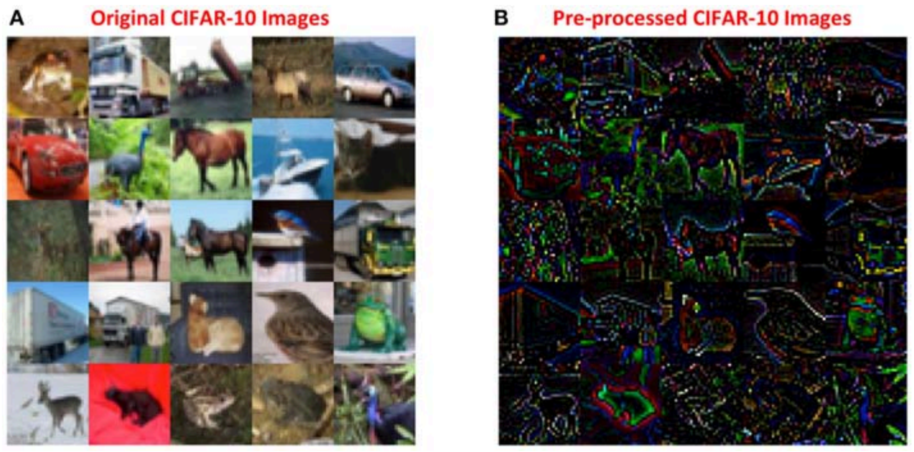


FIGURE 7 | (A) Original 32×32×3 CIFAR-10 images. **(B)** CIFAR-10 images pre-processed using global contrast normalization followed by ZCA whitening (Krizhevsky, 2009).

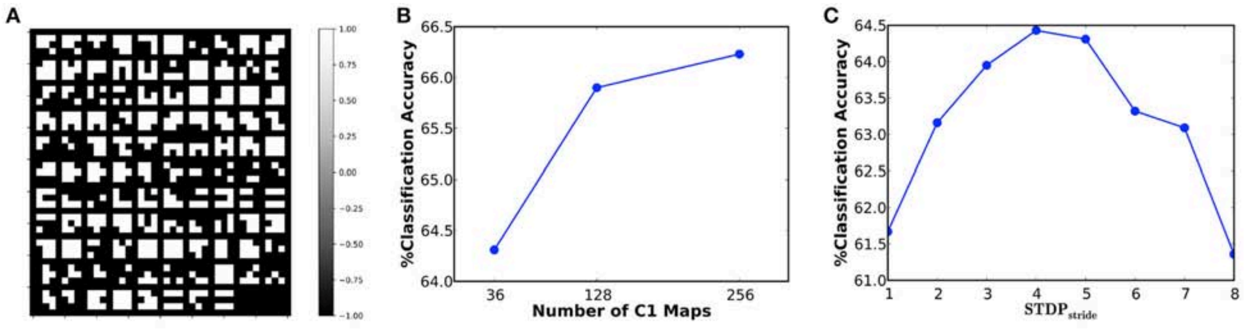


FIGURE 8 | (A) Binary kernels (3×3 in size) of ReStoCNet-1 (36C3-2P-1024FC-10FC ReStoCNet), trained using e/iHB-STDP based unsupervised training methodology, on 5,000 images from the CIFAR-10 dataset. **(B)** Classification accuracy of ReStoCNet vs. the number of convolutional maps. **(C)** Classification accuracy of ReStoCNet-1 vs. the $STDP_{stride}$ used to compute the average spike timing difference of the spiking post-neurons in the convolutional maps.

inference, we simulated the following variants to demonstrate the significance of residual connections for the scalability of deep SNNs:

1. ReStoCNet-3a – This is a variant of ReStoCNet-3 without residual inputs to the third convolutional layer. In addition, the pooled spiking activations of only the third convolutional layer are fed to the fully-connected layer for inference.
2. ReStoCNet-3b – This is a variant of ReStoCNet-3 with residual inputs to the third convolutional layer, wherein the pooled spiking activations of only the third convolutional layer are used for inference.

ReStoCNet-3a, devoid of residual connections, yielded 44.75% accuracy on the CIFAR-10 test set, which is 17.5% lower compared to an accuracy of 62.26% provided by ReStoCNet-3b with residual connections as shown in **Figure 10A**. The higher accuracy of ReStoCNet-3b can be directly attributed to its improved feature learning capability, rendered possible by the residual inputs feeding into the third convolutional layer. The optimal ReStoCNet-3 configuration (with residual connections), wherein the pooled spiking activations of all the convolutional layers are used for inference, offered 65.25% accuracy, which is only comparable to an accuracy of 65.79% provided by ReStoCNet-2 as shown in **Figure 10B**.

Our analysis on ReStoCNet, trained using the e/iHB-STDP based unsupervised training methodology, offers the following key insights. First, it shows that the residual connections are critical for the scalability of deep SNNs. Second, it reveals that the maximum achievable accuracy is limited by the STDP-based unsupervised training methodology as further corroborated by **Figure 11**, which illustrates the unsupervised clustering capability of ReStoCNet-3 for different training images from the CIFAR-10 dataset. In order to visualize the efficiency of unsupervised clustering offered by ReStoCNet-3, we reduce the dimension of the pooled spiking activations of the convolutional layers using Principal Component Analysis (PCA) followed by t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten and Hinton, 2008), and plot the first two t-SNE components for the training images. The t-SNE dimensionality reduction technique computes pair-wise similarities between the data points (images) in the high-dimensional space and projects

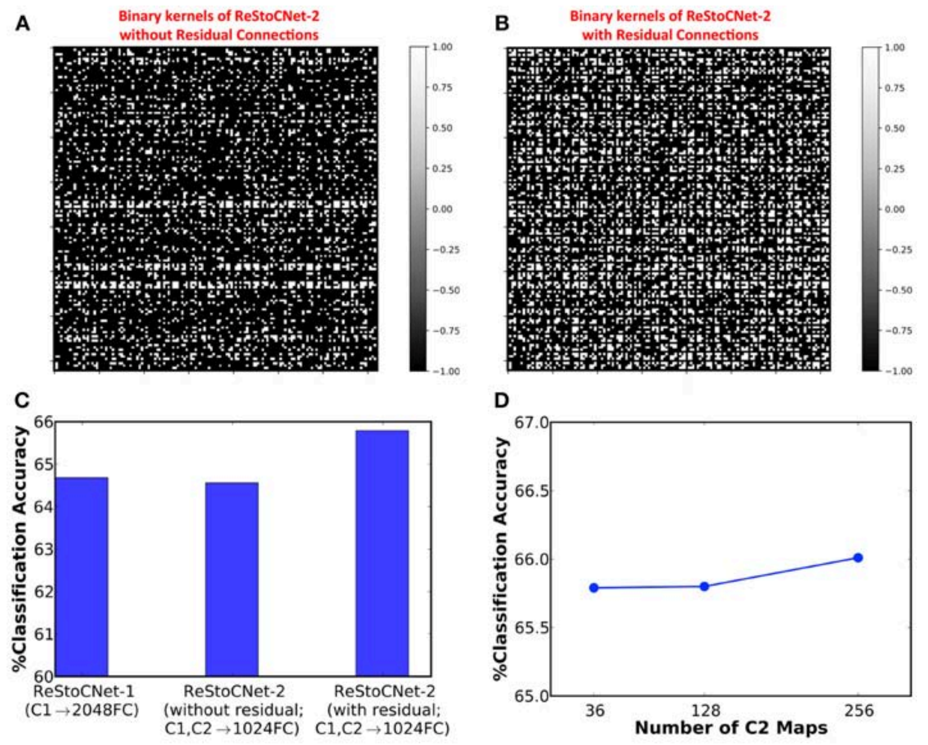


FIGURE 9 | Binary kernels (in second convolutional layer) of ReStoCNet-2 (36C3-36C3-2P-1024FC-10FC ReStoCNet) **(A)** without residual connections, and **(B)** with residual connections from the input to second convolutional layer. **(C)** Classification accuracy of ReStoCNet-2 with and without residual connections compared to that provided by ReStoCNet-1. **(D)** Classification accuracy of ReStoCNet-2 (with residual connections) vs. the number of output maps in the second convolutional layer.

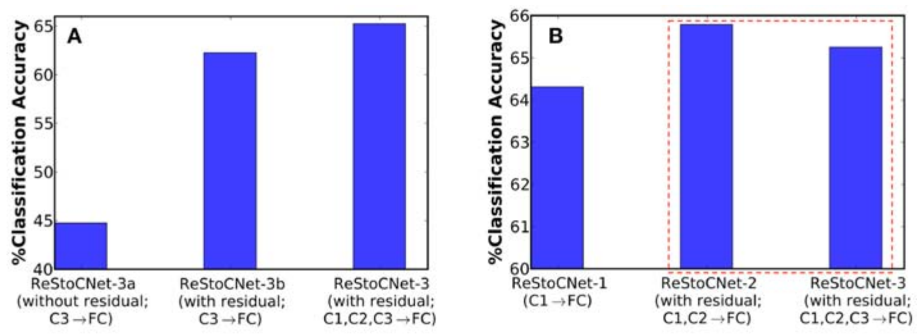


FIGURE 10 | (A) Classification accuracy of three different ReStoCNet-3 (36C3-36C3-36C3-2P-1024FC-10FC) configurations on the CIFAR-10 test set. **(B)** Comparison between the classification accuracy of different ReStoCNet configurations presented in this work.

them to a low-dimensional space that preserves the measured similarities. We refer the readers to Maaten and Hinton (2008) for a review of the t-SNE algorithm for visualizing high-dimensional input data. **Figure 11A** shows the t-SNE scatter plot for 15,000 training images spanning three different classes from the CIFAR-10 dataset, namely, airplane, bird, and frog. The primary objective of any machine learning model is to cluster the images per class together while ensuring sufficient separation among different classes. The t-SNE scatter plot of the pooled spiking activations of ReStoCNet-3 (shown in **Figure 11B**) indicates that, although distinct clusters are formed for the images in each class, there exists considerable overlap among different image clusters.

4. DISCUSSION

4.1. Comparison With Related Works

We compare ReStoCNet with convolutional SNNs, which employ unsupervised training methodology for the convolutional layers and supervised training algorithms like error backpropagation

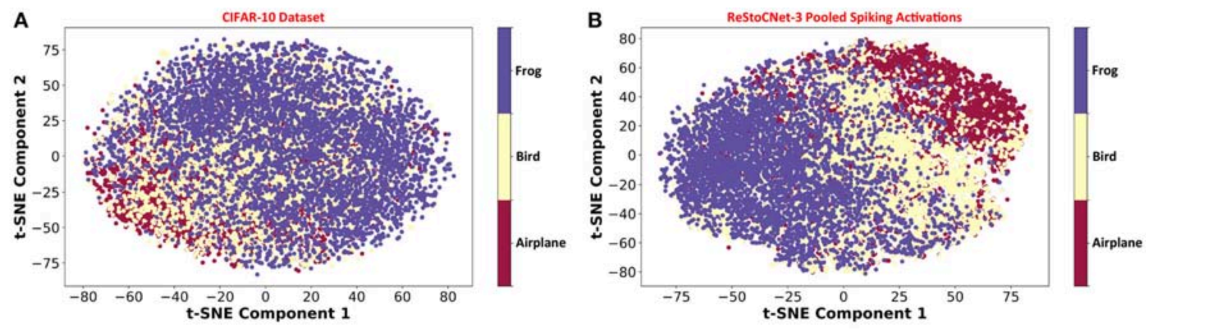


FIGURE 11 | (A) Scatter plot of the first two t-SNE components for 15,000 training images from the CIFAR-10 dataset from the classes: airplane, bird, and frog. **(B)** t-SNE scatter plot of the pooled spiking activations of the convolutional layers in ReStoCNet-3 for the corresponding training images.

for the fully-connected layer, using classification accuracy (on the test set) and kernel memory compression as the evaluation metrics. The memory compression offered by ReStoCNet as a result of using binary kernels in the convolutional layers, referred to as kernel memory compression, is computed as specified by

$$kernel\ memory\ compression = \frac{N_{baseline} \times ksize_{baseline} \times ksize_{baseline} \times nbits_{full\_precision}}{N_{ReStoCNet} \times ksize_{ReStoCNet} \times ksize_{ReStoCNet} \times nbits_{binary}} \tag{8}$$

where $N_{ReStoCNet}$ ($N_{baseline}$) and $ksize_{ReStoCNet}$ ($ksize_{baseline}$) are the number of kernels and kernel size, respectively, in ReStoCNet (baseline convolutional SNN used for comparison), and $nbits_{binary}$ and $nbits_{full\_precision}$ are the hardware bit-precision required for storing the binary and full-precision kernels, which are set to 2-bits and 32-bits, respectively. Note that the binary kernels in ReStoCNet require storage capacity of 2-bits per synaptic weight since they are constrained to binary states -1 and $+1$. **Table 5** shows that the classification accuracy offered by ReStoCNet for MNIST digit recognition is comparable to that reported for convolutional SNNs composed of full-precision kernels trained using unsupervised learning methodologies. Specifically, a 36C3-2P-128FC-10FC ReStoCNet offers 98.54% accuracy on the MNIST test set, which compares favorably with that (98.36%) provided by the convolutional SNN presented in Tavanaei and Maida (2017), composed of single convolutional layer with 32 maps and 5×5 full-precision kernels trained using STDP. The proposed ReStoCNet offers 39.5× kernel memory compression by virtue of using smaller 3×3 binary kernels under iso-accuracy conditions for MNIST digit recognition. On the contrary, very few works have benchmarked convolutional SNNs, trained using unsupervised learning algorithms, on the CIFAR-10 dataset. Panda and Roy (2016) proposed spike-based convolutional Auto-Encoders, where the kernels in every convolutional layer are trained in an unsupervised manner using error backpropagation to regenerate the input spike patterns. Ferré et al. (2018) presented convolutional SNN (without residual connections), where the kernels are trained using a simple Hebbian STDP learning rule. **Table 6** shows that ReStoCNet provides 4–5% lower accuracy than that reported in both the related works. In particular, a 256C3-2P-1024FC-10FC ReStoCNet yields 4.97% lower accuracy than that provided by the 64C7-8P-512FC-512FC-10FC convolutional SNN (Ferré et al., 2018) while offering 21.7× kernel memory compression. Note that the convolutional SNN presented in Ferré et al. (2018) is simulated by single-step forward propagation using input rates while ReStoCNet is simulated using input spike trains over multiple time-steps.

Finally, we note that deep learning Binary Neural Networks (BNNs) (Courbariaux et al., 2015; Rastegari et al., 2016; Hubara et al., 2017), which use binary activations for the neurons in every layer except the input and output layers and binary weights, have been demonstrated to yield superior classification accuracy than that provided by ReStoCNet. Nevertheless, ReStoCNet offers the following advantages over BNNs. First, ReStoCNet is inherently suited for processing spatiotemporal spike trains from event-based audio and vision sensors as shown by Stromatias et al. (2017) for convolutional SNNs with full-precision weights since it computes with static image pixels mapped to spike trains. BNNs, on the contrary, use real-valued pixel intensities for the input layer. Second, ReStoCNet is amenable for efficient implementation in event-driven asynchronous neuromorphic hardware platforms like IBM *TrueNorth* (Merolla et al., 2014) and Intel *Loihi* (Davies et al., 2018) since it uses {0, 1} for the outputs of the spiking neurons in every convolutional layer. The weighted sum of the input spikes with the synaptic weights in the convolutional layers needs to be computed only in the event of a spike fired by the corresponding input neurons. In addition, only the sparse spiking events need to be transmitted between the layers. The event-driven computing capability offered by ReStoCNet can be exploited to achieve higher energy efficiency in neuromorphic hardware implementations by minimizing the computation and communication energy in the absence of spiking events. BNNs, on the other hand, use {1, -1} for the neuronal activations and either {1, -1} (Courbariaux et al., 2015) or {α, $-\alpha$} (Rastegari et al., 2016) where α is a layer-wise scaling factor for the weights to achieve good accuracy and stable training convergence (Pfeiffer and Pfeil, 2018). Hence, the computation of the weighted input sum and communication of the binarized neuronal activations need to be carried out for all the neurons in every layer in a synchronous manner, which is in contrast to the event-based asynchronous computing capability provided by ReStoCNet.

TABLE 5 | Classification accuracy of SNN models, which use unsupervised training methodology for the hidden/convolutional layers and supervised training algorithm for the output (classification) layer, on the MNIST test set.

Model	Size	Training methodology	Accuracy (%)
FC_SNN (Yousefzadeh et al., 2018)	6400FC-10FC	Probabilistic STDP + ANN backpropagation	95.70
ConvSNN (Panda and Roy, 2016)	12C5-2P-64C5-2P-10FC	SNN backpropagation	99.08
ConvSNN (Stromatias et al., 2017)	18C7-2P-10FC	Fixed Gabor kernels + ANN backpropagation	98.20
ConvSNN (Lee et al., 2018b)	16C3-16C3-2P-10FC	STDP	91.10
ConvSNN (Ferré et al., 2018)	8C5-2P-16C5-2P-120FC-60FC-10FC	STDP + ANN backpropagation	98.49
ConvSNN (Kheradpisheh et al., 2018)	30C5-2P-100C5-2P-10FC	STDP + Support Vector Machine	98.40
ConvSNN (Tavanaei et al., 2018)	64C5-2P-1500FC-10FC	STDP	98.61
ConvSNN (Mozafari et al., 2018)	30C5-2P-250C3-3P-200C5-5P	Reward-modulated STDP	97.20
ConvSNN (Tavanaei and Maida, 2017)	32C5-2P-128FC-10FC	STDP + Support Vector Machine	98.36
ReStoCNet (our work)	36C3-2P-128FC-10FC	Probabilistic eHB-STDP + ANN backpropagation	98.54

TABLE 6 | Classification accuracy of SNN models, which use unsupervised training methodology for the hidden/convolutional layers and supervised training algorithm for the output (classification) layer, on the CIFAR-10 test set.

Model	Size	Training methodology	Accuracy (%)
ConvSNN (Panda and Roy, 2016)	32C5-2P-32C5-2P-64C4-10FC	SNN backpropagation	70.16
ConvSNN (Ferré et al., 2018)	64C7-8P-512FC-512FC-10FC	STDP + ANN backpropagation	71.20
ReStoCNet (our work)	256C3-2P-1024FC-10FC	Probabilistic e/iHB-STDP + ANN backpropagation	66.23

Last, ReStoCNet offers a memory-efficient solution for enabling on-chip intelligence in resource-constrained battery-powered Internet of Things (IoT) edge devices since the binary kernels are trained using probabilistic-STDP based local learning rule that can be efficiently implemented on-chip. Learning is achieved by probabilistically switching the binary kernel weights between the allowed states based on spike timing, which precludes the need for storing the full-precision weights and enhances the memory efficiency during training. BNNs, on the other hand, are trained using error backpropagation algorithms that update the full-precision weights based on the backpropagated error gradients and binarize the modified weights for forward propagation and computing the error gradients. Thus, ReStoCNet provides a promising alternative for energy- and memory-efficient computing during both training and inference in IoT edge devices, for instance, surveillance cameras, which produce large volumes of real-time data. It is inefficient for these devices to continuously offload raw/compressed data to the cloud for training. This is because the sheer volume of generated data could exceed the bandwidth available for transmitting them to the cloud. Alternatively, there could be connectivity issues restricting communication between the edge and the cloud. In addition, there are also security and data privacy issues that need to be addressed while sending (receiving) data to (from) the cloud. Hence, it is highly desirable to equip the edge devices with on-chip intelligence so that they can learn from real-time input data and invoke the cloud occasionally to update the on-chip trained weights using more complex algorithms. The proposed approach is also suited for building intelligent autonomous systems like robots and self-flying drones. For example, it is beneficial to embed on-chip learning in autonomous robots used for disaster relief operations that enables them to navigate obstacles and scour the disaster site for survivors. In the instance of self-flying drones used for reconnaissance operations, on-chip intelligence can enable them to effectively navigate the enemy territory and improve the chances of a successful mission.

The classification accuracy of ReStoCNet for complex applications could be improved by augmenting the layer-wise unsupervised training methodology with a global supervised training mechanism. Recent works have proposed error backpropagation algorithms for the supervised training of SNNs (Lee et al., 2016, 2018a; Panda and Roy, 2016; Jin et al., 2018; Mostafa, 2018; Wu et al., 2018). However, the backpropagation algorithms for SNNs, some of which backpropagate errors at multiple time-steps, are computationally prohibitive and prone to unstable convergence behaviors (Lee et al., 2018a). In this regard, Neftci et al. (2017) proposed event-driven random backpropagation that prevents the need for calculating and backpropagating precise error gradients. Future works could explore a hybrid unsupervised (local) and supervised (global) training methodology for ReStoCNet to obtain favorable trade-offs between classification accuracy and training effort as was shown by Lee et al. (2018a) for full-precision convolutional SNNs without residual connections. Such a hybrid approach would also preclude the need for using the pooled spiking activations of all the convolutional layers for inference, thereby enhancing the scalability of deep ReStoCNets.

4.2. Applicability of ReStoCNet for Neuromorphic Hardware Implementations

Together with research efforts that are geared toward the exploration of bio-plausible SNN algorithms (architectures and learning methodologies), parallel efforts are underway to develop neuromorphic hardware implementations with on-chip intelligence, which can exploit the inherent computational efficiency offered by the SNN algorithms. IBM *TrueNorth* (Merolla et al., 2014) and Intel *Loihi* (Davies et al., 2018) are recent demonstrations of event-driven neuromorphic hardware that were realized using the conventional CMOS technology. CMOS-based neuromorphic hardware implementations are area- and power-intensive because of the mismatch between the spiking neuronal/synaptic circuits and the neuroscience processes governing their dynamics. In this regard, nanoelectronic devices such as Ag-Si memristor (Jo et al., 2010), Phase-Change Memory (PCM) (Suri et al., 2011), Resistive Random Access Memory (Rajendran et al., 2013) and domain-wall Magnetic Tunnel Junctions (MTJs) (Sengupta et al., 2016a) that are capable of naturally mimicking multilevel synaptic dynamics have been proposed as potential candidates for achieving improved energy efficiency compared to CMOS-only realizations. However, as the technology is scaled, the multilevel memristive and spintronic devices suffer from limited bit-precision and exhibit stochastic behavior in the presence of thermal noise. The proposed ReStoCNet, which is composed of binary kernels trained using probabilistic HB-STDP, is naturally suited for neuromorphic hardware implementations based on stochastic device technologies as elaborated in the following paragraph.

Stochastic device technologies such as Conductive-Bridge Random Access Memory (CBRAM) (Suri et al., 2013), RRAM (Kavehei and Skafidas, 2014), MTJ (Vincent et al., 2015; Sengupta et al., 2016b; Srinivasan et al., 2016), and PCM (Tuma et al., 2016) have been shown to efficiently implement stochastic neuronal and synaptic models. The intrinsic stochastic switching behavior of these devices can be exploited to realize the probabilistic switching of a binary synapse during training without the need for costly random number generators to implement the stochastic operations as illustrated with MTJ-based synapse. An MTJ is composed of two ferromagnetic layers, namely, a pinned layer whose magnetization is fixed and a free layer whose magnetization can be switched, separated by a tunneling oxide barrier. It exhibits two stable conductance states based on the relative orientation of the pinned layer and free layer magnetizations, which can be switched probabilistically by passing charge current through a Heavy Metal (HM) located underneath the MTJ structure. Srinivasan et al. (2016) showed that the MTJ-HM heterostructure, with independent spike-transmission and programming current paths, can efficiently realize a stochastic binary synapse. During training, the MTJ is switched probabilistically based on the time difference between pre- and post-spikes by passing the appropriate current through the HM. During inference, an input pre-spike gets modulated with the trained MTJ conductance to produce resultant current into the post-neuron. Srinivasan et al. (2016) also presented peripheral circuits required to implement an exponential probabilistic-STDP rule, which needs to be modified for realizing the proposed HB-STDP rule. We note that CBRAM, RRAM, and PCM devices can similarly be used to realize a stochastic binary synapse during training by modulating the input voltage based on spike timing (Suri et al., 2013; Kavehei and Skafidas, 2014). Crossbar-based hardware implementations based on these stochastic device technologies with on-chip learning capability have been demonstrated for efficiently realizing binary fully-connected SNNs (Suri et al., 2013; Srinivasan et al., 2016), which consists of a unique synaptic weight connecting every pair of pre- and post-neurons. Recently Wijesinghe et al. (2018) showed that weight-shared convolutional SNNs such as ReStoCNet can be mapped to crossbar-based hardware implementations. However, large-scale networks with increased number of neurons and synapses cannot be mapped to a single large crossbar due to non-idealities that could result in erroneous computations. Hardware architectures composed of multiple smaller crossbars can be used to efficiently realize large-scale networks (Shafiee et al., 2016; Ankit et al., 2017; Song et al., 2017). Finally, we note that the fully-connected classification layer in ReStoCNet, which is composed of artificial ReLU neurons, cannot be directly implemented in event-driven asynchronous neuromorphic hardware platforms. The fully-connected layer of ReLU neurons could be mapped to Integrate-and-Fire neurons post training for inference within the neuromorphic fabric as shown by Diehl et al. (2015). Alternatively, fully-connected layer of Leaky-Integrate-and-Fire neurons can be trained using spike-based backpropagation algorithms for training and/or inference within the neuromorphic fabric.

5. CONCLUSION

In this work, we proposed ReStoCNet, a residual stochastic multilayer convolutional SNN composed of binary kernels, for memory-efficient neuromorphic computing. We presented probabilistic Hybrid-STDP (HB-STDP) learning rule, integrating Hebbian and anti-Hebbian learning mechanisms, for training the binary kernels constituting ReStoCNet in a layer-wise unsupervised manner. We demonstrated up to 3-layer deep ReStoCNet and showed that residual connections are critical to enabling the deeper convolutional layers to self-learn useful high-level input features and improving the scalability of deep SNNs. ReStoCNet offered 98.54% accuracy and 39.5× kernel memory compression compared to full-precision (32-bit) convolutional SNN under iso-accuracy conditions for MNIST digit recognition.

On the CIFAR-10 dataset, ReStoCNet provided 66.23% accuracy and 21.7× kernel memory compression, albeit with 5% accuracy degradation compared to full-precision convolutional SNN. We believe that ReStoCNet, with event-driven computing capability and memory-efficient probabilistic learning with binary kernels, is ideally suited for neuromorphic hardware implementations based on CMOS and stochastic emerging device technologies like Resistive Random Access Memory, Phase-Change Memory, and Magnetic Tunnel Junctions that can potentially lead to much improved energy efficiency in battery-powered IoT edge devices.

DATA AVAILABILITY

Publicly available datasets were analyzed in this study. This data can be found here: https://www.cs.toronto.edu/~kriz/cifar.html.

AUTHOR CONTRIBUTIONS

GS wrote the paper and performed the simulations. All authors helped with developing the concepts, conceiving the experiments, and writing the paper.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by the Semiconductor Research Corporation, the National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship, and by the U.S. Army Research Laboratory and the U.K. Ministry of Defense under Agreement Number W911NF-16-3-0001.

REFERENCES

Ankit, A., Sengupta, A., Panda, P., and Roy, K. (2017). "Resparc: a reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017* (Austin, TX: ACM), 27.

Bi, G.-Q., and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). "Binaryconnect: training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems* (Montréal, QC), 3123–3131.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. F. (2001). *Theoretical Neuroscience*, Vol. 806. Cambridge, MA: MIT Press.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Neural Networks (IJCNN), 2015 International Joint Conference on* (Killarney: IEEE), 1–8.

Ferré, P., Mamalet, F., and Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based stdp. *Front. Comput. Neurosci.* 12:24. doi: 10.3389/fncom.2018.00024

Goodman, D. F., and Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Front. Neuroinformatics* 2:5. doi: 10.3389/neuro.11.005.2008

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778.

Hebb, D. (1949). The organization of behavior. New York, NY: Wiley.

Hu, Y., Tang, H., Wang, Y., and Pan, G. (2018). Spiking deep residual network. arXiv:1805.01352v1.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 6869–6898. Available online at: http://jmlr.org/papers/v18/16-456.html

Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). "Spatial transformer networks," in *Advances in Neural Information Processing Systems* (Montreal, QC), 2017–2025.

Jin, Y., Zhang, W., and Li, P. (2018). "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems* (Montréal, QC), 7005–7015.

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* 10, 1297–1301. doi: 10.1021/nl904092h

Jug, F. (2012). *On Competition and Learning in Cortical Structures*. Doctoral thesis, ETH Zurich.

Kavehei, O., and Skafidas, E. (2014). "Highly scalable neuromorphic hardware with 1-bit stochastic nano-synapses," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on* (Melbourne, VIC: IEEE), 1648–1651. doi: 10.1109/ISCAS.2014.6865468

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. arXiv:1412.6980.

Krizhevsky, A., and Hinton, G. (2009). *Learning Multiple Layers of Features From Tiny Images*, Vol. 1. Technical report, University of Toronto, 7.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018a). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2018b). "Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity," in *IEEE Trans. Cogn. Dev. Syst.* doi: 10.1109/TCDS.2018.2833071. [Epub ahead of print].

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Lowel, S., and Singer, W. (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science* 255, 209–212. doi: 10.1126/science.1372754

Maaten, L. v. d., and Hinton, G. (2008). Visualizing data using t-sne. *J. Mach. Learn. Res.* 9, 2579–2605. Available online at: http://www.jmlr.org/papers/v9/vandermaaten08a.html

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e31. doi: 10.1371/journal.pcbi.0030031

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mostafa, H. (2018). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2018). Combining stdp and reward-modulated stdp in deep convolutional spiking neural networks for digit recognition. arXiv: 1804.00227.

Nair, V., and Hinton, G. E. (2010). "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa), 807–814.

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324

Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 299–306.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in pytorch," in *NIPS Workshop* (Long Beach, CA).

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Querlioz, D., Bichler, O., Vincent, A. F., and Gamrat, C. (2015). Bioinspired programming of memory devices for implementing an inference engine. *Proc. IEEE* 103, 1398–1416. doi: 10.1109/JPROC.2015.2437616

Rajendran, B., Liu, Y., Seo, J.-S., Gopalakrishnan, K., Chang, L., Friedman, D. J., et al. (2013). Specifications of nanoscale devices and circuits for neuromorphic computational systems. *IEEE Trans. Electron Devices* 60, 246–253. doi: 10.1109/TED.2012.2227969

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). "Xnor-net: imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision* (Amsterdam: Springer), 525–542.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323:533. doi: 10.1038/323533a0

Sengupta, A., Banerjee, A., and Roy, K. (2016a). Hybrid spintronic-cmos spiking neural network with on-chip learning: devices, circuits, and systems. *Phys. Rev. Appl.* 6:064003. doi: 10.1103/PhysRevApplied.6.064003

Sengupta, A., Panda, P., Wijesinghe, P., Kim, Y., and Roy, K. (2016b). Magnetic tunnel junction mimics stochastic cortical spiking neurons. *Sci. Rep.* 6:30039. doi: 10.1038/srep30039

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). Isaac: a convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars. *ACM SIGARCH Comput. Architect. News* 44, 14–26. doi: 10.1145/3007787.3001139

Song, L., Qian, X., Li, H., and Chen, Y. (2017). "Pipelayer: a pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on* (Austin, TX: IEEE), 541–552.

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3:919. doi: 10.1038/78829

Srinivasan, G., Panda, P., and Roy, K. (2018). Stdp-based unsupervised feature learning using convolution-over-time in spiking neural networks for energy-efficient neuromorphic computing. *J. Emerg. Technol. Comput. Syst.* 14, 44:1–44:12. doi: 10.1145/3266229

Srinivasan, G., Sengupta, A., and Roy, K. (2016). Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip stdp learning. *Sci. Rep.* 6:29545. doi: 10.1038/srep29545

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1929–1958.

Stromatias, E., Soto, M., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017). An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Front. Neurosci.* 11:350. doi: 10.3389/fnins.2017.00350

Suri, M., Bichler, O., Querlioz, D., Cueto, O., Perniola, L., Sousa, V., et al. (2011). "Phase change memory as synapse for ultra-dense neuromorphic systems: application to complex visual pattern extraction," in *2011 IEEE International Electron Devices Meeting (IEDM)*, (Washington, DC: IEEE), 4.

Suri, M., Querlioz, D., Bichler, O., Palma, G., Vianello, E., Vuillaume, D., et al. (2013). Bio-inspired stochastic computing using binary cbram synapses. *IEEE Trans. Electron Devices* 60, 2402–2409. doi: 10.1109/TED.2013.2263000

Tavanaei, A., Kirby, Z., and Maida, A. S. (2018). "Training spiking convnets by stdp and gradient descent," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro), 1–8.

Tavanaei, A., and Maida, A. S. (2017). "Multi-layer unsupervised learning in a spiking convolutional neural network," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 2023–2030.

Thiele, J. C., Bichler, O., and Dupret, A. (2018). Event-based, timescale invariant unsupervised online deep learning with STDP. *Front. Comput. Neurosci.* 12:46. doi: 10.3389/fncom.2018.00046

Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A., and Eleftheriou, E. (2016). Stochastic phase-change neurons. *Nat. Nanotechnol.* 11, 693–699. doi: 10.1038/nnano.2016.70

Vincent, A. F., Larroque, J., Locatelli, N., Romdhane, N. B., Bichler, O., Gamrat, C., et al. (2015). Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *IEEE Trans. Biomed. Circ. Syst.* 9, 166–174. doi: 10.1109/TBCAS.2015.2414423

Wijesinghe, P., Ankit, A., Sengupta, A., and Roy, K. (2018). An all-memristor deep spiking neural computing system: a step toward realizing the low-power stochastic brain. *IEEE Trans. Emerging Top. Comput. Intell.* 2, 345–358. doi: 10.1109/TETCI.2018.2829924

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Yousefzadeh, A., Stromatias, E., Soto, M., Serrano-Gotarredona, T., and Linares-Barranco, B. (2018). On practical issues for stochastic stdp hardware with 1-bit synaptic weights. *Front. Neurosci.* 12:665. doi: 10.3389/fnins.2018.00665

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 27 March 2019
doi: 10.3389/fnins.2019.00252

A Delay Learning Algorithm Based on Spike Train Kernels for Spiking Neurons

Xiangwen Wang, Xianghong Lin and Xiaochao Dang*

College of Computer Science and Engineering, Northwest Normal University, Lanzhou, China

Neuroscience research confirms that the synaptic delays are not constant, but can be modulated. This paper proposes a supervised delay learning algorithm for spiking neurons with temporal encoding, in which both the weight and delay of a synaptic connection can be adjusted to enhance the learning performance. The proposed algorithm firstly defines spike train kernels to transform discrete spike trains during the learning phase into continuous analog signals so that common mathematical operations can be performed on them, and then deduces the supervised learning rules of synaptic weights and delays by gradient descent method. The proposed algorithm is successfully applied to various spike train learning tasks, and the effects of parameters of synaptic delays are analyzed in detail. Experimental results show that the network with dynamic delays achieves higher learning accuracy and less learning epochs than the network with static delays. The delay learning algorithm is further validated on a practical example of an image classification problem. The results again show that it can achieve a good classification performance with a proper receptive field. Therefore, the synaptic delay learning is significant for practical applications and theoretical researches of spiking neural networks.

Keywords: spiking neural networks, supervised learning, spike train kernels, delay learning, synaptic delays

OPEN ACCESS

Edited by:
Yansong Chua,
*Institute for Infocomm Research (A*STAR), Singapore*

Reviewed by:
Shaista Hussain,
*Institute of High Performance Computing (A*STAR), Singapore*
Liam P. Maguire,
Ulster University, United Kingdom

****Correspondence:***
Xianghong Lin
linxh@nwnu.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *08 October 2018*
Accepted: *04 March 2018*
Published: *27 March 2019*

Citation:
Wang X, Lin X and Dang X (2019) A Delay Learning Algorithm Based on Spike Train Kernels for Spiking Neurons. Front. Neurosci. 13:252. doi: 10.3389/fnins.2019.00252

1. INTRODUCTION

Spiking neural networks (SNNs) that composed of biologically plausible spiking neurons are usually known as the third generation of artificial neural networks (ANNs) (Maass, 1997). The spike trains are used to represent and process the neural information in spiking neurons, which can integrate many aspects of neural information, such as time, space, frequency, and phase, etc. (Whalley, 2013; Walter et al., 2016). As a new brain-inspired computational model of the neural network, SNN has more powerful computing power compared with a traditional neural network model (Maass, 1996). SNNs can simulate all kinds of neural signals and arbitrary continuous functions, which are very suitable for processing the brain neural signals (Ghosh-Dastidar and Adeli, 2009; Beyeler et al., 2013; Gütig, 2014).

Supervised learning for SNNs refers to that for multiple given input spike trains and desired output spike trains, finding an appropriate synaptic weight matrix of the SNNs in order to assimilate the actual output spike trains of output neurons to the corresponding desired output spike trains, that is, the value of the error evaluation function between them is the smallest. Researchers have proposed many supervised multi-spike learning algorithms for spiking neurons in recent years (Lin et al., 2015b). The basic ideas of these algorithms mainly include gradient descent, synaptic plasticity, and spike train convolution.

Supervised learning algorithms based on gradient descent use gradient computation and error back-propagation for adjusting the synaptic weights, and ultimately minimize the error function that indicates the deviation between the actual and desired output spike trains. Xu et al. (2017) proposed a supervised learning algorithm for spiking neurons based on gradient descent, in which an online adjustment mechanism is used. The basic idea of supervised learning algorithms based on synaptic plasticity is using the mechanism of synaptic plasticity caused by the timing correlation of spike trains of presynaptic and postsynaptic neurons to design the supervised learning rules. Representative algorithms are the remote supervised method (ReSuMe) (Ponulak and Kasiński, 2010) and its extensions (Lin et al., 2016, 2018). Supervised learning algorithms based on spike train convolution are constructed by the inner products of spike trains (Paiva et al., 2009; Park et al., 2013). Discrete spike trains are firstly converted to continuous functions through the convolution calculation of the specific kernel function, and then constructing the supervised learning algorithm for SNNs. The adjustment of synaptic weights depends on the convolved continuous functions corresponding to spike trains, which can realize the learning of the spatio-temporal pattern of the spike trains. Representative algorithms are spike pattern association neuron (SPAN) (Mohemmed et al., 2012), precise-spike-driven (PSD) (Yu et al., 2013), and the work of Lin et al. (Lin et al., 2015a; Wang et al., 2016; Lin and Shi, 2018).

Experimental research (Minneci et al., 2012) proves that synaptic delays widely exist in biological neural networks. The time delay has an effect on the processing ability of the nervous system (Xu et al., 2013). At present, in most supervised learning algorithms for SNNs, only the connection strength, namely the synaptic weight between pre- and post-synapse, is adjusted. Neuroscientific studies have shown that the synaptic delays in the biological nervous system are not always invariant, but can be modulated (Lin and Faber, 2002; Boudkkazi et al., 2011). However, efficient synaptic delay learning algorithms are few. In recent years, researchers have introduced the delay learning to ReSuMe learning rule (Ponulak and Kasiński, 2010) and proposed some ReSuMe-based delay learning algorithms (Taherkhani et al., 2015a,b, 2018; Guo et al., 2017). Simulation results show that the delay versions of ReSuMe achieve learning accuracy and learning speed improvements compared with the original ReSuMe. Shrestha et al. (Shrestha and Song, 2016) formulated an adaptive learning rate scheme for delay adaptation in the SpikeProp algorithm (Bohte et al., 2002) based on delay convergence analysis. Simulation results of spike train learning show that the extended algorithm improves learning performance of the basic SpikeProp algorithm. There are also some other delay learning algorithms (Napp-Zinn et al., 1996; Wang et al., 2012; Hussain et al., 2014) have been proposed, and further implemented by hardware.

In this paper, we propose a new supervised delay learning algorithm based on spike train kernels for spiking neurons, in which both the synaptic weights and the synaptic delays can be adjusted. The rest of this paper is organized as follows. In section 2, we first introduce the spiking neuron model and the kernel representation of the spike train used in this paper and then derive the supervised learning rules of both synaptic weights and synaptic delays using gradient descent method. A series of spike train learning tasks and an image classification task are performed to test and verify the learning performance of our proposed learning algorithm in section 3. The discussion of our proposed algorithm is presented in section 4. Finally, we conclude this paper in section 5.

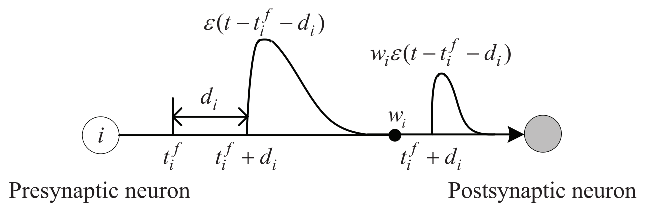


FIGURE 1 | Spike response function.

2. MATERIALS AND METHODS

2.1. Spiking Neuron and Spike Train Representation

2.1.1. Spike Response Model

The short-term memory spike response model (SRM) (Gerstner and Kistler, 2002) is employed in delay learning. It expresses the membrane potential u at time t as an integral over the past, including a model of refractoriness. In the short-term memory SRM, only the last fired spike t_o^l contributes to the refractoriness. Assuming that a neuron has N_I input synapses, the ith synapse transmits a total of N_i spikes and the fth spike ($f \in [1, N_i]$) is fired at time t_i^f. The internal state $u(t)$ of the neuron at time t is given by:

$$u(t) = \sum_{i=1}^{N_I}\sum_{f=1}^{N_i} w_i\varepsilon(t - t_i^f - d_i) + \eta(t - t_o^l) \tag{1}$$

where w_i and d_i are the synaptic weight and the synaptic delay for the ith synapse, respectively. When the internal state variable $u(t)$ crosses the firing threshold θ, the neuron fires a spike.

The spike response function $\varepsilon(t - t_i^f - d_i)$ describes the effect of the presynaptic spike on the internal state of the postsynaptic neuron, as shown in **Figure 1**. It is expressed as:

$$\varepsilon(t - t_i^f - d_i) = \begin{cases} \frac{t-t_i^f-d_i}{\tau}\exp(1 - \frac{t-t_i^f-d_i}{\tau}) & , t - t_i^f - d_i > 0 \\ 0 & , t - t_i^f - d_i \le 0 \end{cases} \tag{2}$$

where τ indicates the time decay constant of postsynaptic potentials, which determines the shape of the spike response function.

In addition, $\eta(t - t_o^l)$ is the refractoriness function, which is mainly reflected in the effect that only the last output spike t_o^l

contributes to the refractoriness:

$$\eta(t - t_o^l) = \begin{cases} -\theta \exp(-\frac{t-t_o^l}{\tau_R}) & , t - t_o^l > 0 \\ 0 & , t - t_o^l \leq 0 \end{cases} \tag{3}$$

where θ is the neuron threshold. τ_R is the time constant, which determines the shape of refractoriness function. When $t - t_o^l \in (0, \infty)$, the refractoriness function $\eta(t - t_o^l)$ is negative. When $t - t_o^l \to 0$, the minimum value of $\eta(t - t_o^l)$ is $-\theta$. When $t - t_o^l \to \infty$, the value of $\eta(t - t_o^l)$ is gradually increased to 0.

2.1.2. Spike Train and Its Kernel Representation

The spike train $s = \{t^f \in \Gamma : f = 1, \cdots, N\}$ represents the ordered sequence of spike times fired by the spiking neuron in the time interval $\Gamma = [0, T]$, and can be expressed formally as:

$$s(t) = \sum_{f=1}^{N} \delta\left(t - t^f\right) \tag{4}$$

where t^f is the fth spike time in $s(t)$, N is the number of spikes in $s(t)$, and $\delta(\cdot)$ represents the Dirac delta function, $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. Considering the synaptic delay in the input spike train, the input spike train $s_i(t - d_i)$ with synaptic delay is defined as:

$$s_i(t - d_i) = \sum_{f=1}^{N_i} \delta\left(t - t_i^f - d_i\right) \tag{5}$$

where t_i^f is the fth spike in the input spike train $s_i(t - d_i)$, d_i is the synaptic delay between presynaptic neuron i and postsynaptic neuron, and N_i is the number of spikes in $s_i(t - d_i)$.

In order to facilitate the analysis and calculation, we can choose a specific kernel function $\kappa(\cdot)$, using the convolution to convert the discrete spike train to a continuous function:

$$f_s(t) = s(t) * \kappa(t) = \sum_{f=1}^{N} \kappa\left(t - t^f\right) \tag{6}$$

Therefore, the convolved continuous functions corresponding to the input spike train $s_i(t - d_i)$, actual output spike train $s_o(t)$, and desired output spike train $s_d(t)$ can be expressed as follows according to Equation (6):

$$f_{s_i}(t - d_i) = s_i(t - d_i) * \kappa(t) = \sum_{f=1}^{N_i} \kappa\left(t - t_i^f - d_i\right) \tag{7}$$

$$f_{s_o}(t) = s_o(t) * \kappa(t) = \sum_{h=1}^{N_o} \kappa\left(t - t_o^h\right) \tag{8}$$

$$f_{s_d}(t) = s_d(t) * \kappa(t) = \sum_{g=1}^{N_d} \kappa\left(t - t_d^g\right) \tag{9}$$

where t_i^f, t_o^h, and t_d^g are spikes in $s_i(t - d_i)$, $s_o(t)$, and $s_d(t)$, respectively. N_i, N_o, and N_d are numbers of spikes in $s_i(t - d_i)$, $s_o(t)$, and $s_d(t)$, respectively.

In SNNs, neural information or external stimuli is encoded into spike trains. The computation performed by a single spiking neuron can be defined as a mapping from the presynaptic spike trains to the appropriate postsynaptic spike train. In order to analyze the relationship between the presynaptic and postsynaptic spike trains, we use linear-nonlinear Poisson (LNP) model (Schwartz et al., 2006), in which the spiking activity of the postsynaptic neuron is defined by the estimated intensity functions of the presynaptic neurons. Some researches show that the relationship between the postsynaptic spike train $s_o(t)$ and the contributions of all presynaptic spike trains $s_i(t - d_i)$ can be expressed as a linear relationship for excitatory synapse through the convolved continuous functions (Cash and Yuste, 1999; Carnell and Richardson, 2005):

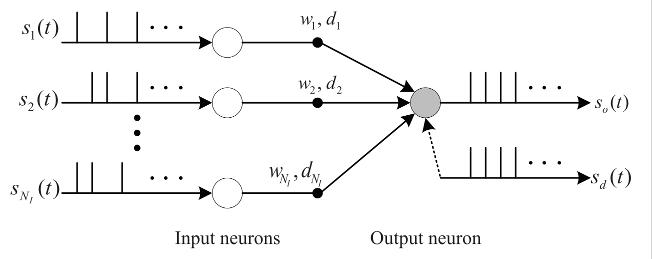


FIGURE 2 | Network structure of neurons with synaptic delays.

$$f_{s_o}(t) = \sum_{i=1}^{N_I} w_i f_{s_i}(t - d_i) \tag{10}$$

where w_i represents the synaptic weight between the presynaptic neuron i and the postsynaptic neuron, and N_I is the number of presynaptic neurons.

2.2. Learning Rules Based on Spike Train Kernels

In this section, we use the gradient descent method to deduce the learning rule of synaptic weights and delays. We consider a fully connected feed-forward network structure of spiking neurons as shown in **Figure 2**. There are N_I input neurons and one output neuron in this model. There is only one synaptic connection between an input neuron and an output neuron. Each synapse has a connection weight w_i and a time delay d_i. The aim of the delay learning method is to train the neuron to produce a desired output spike train $s_d(t)$ in response to multiple spatio-temporal input spike patterns $s_i(t - d_i)$. In the synaptic delay learning model, both the synaptic weight w_i and the synaptic delay d_i are adjusted to train the output neuron to fire the actual output spike train $s_o(t)$ toward the desired output spike train $s_d(t)$.

Defining the error function of the network is an important prerequisite for supervised learning of spiking neurons. The instantaneous error for the network can be formally defined in terms of the square difference between the convolved continuous functions $f_{s_o}(t)$ and $f_{s_d}(t)$ corresponding to the actual output spike train $s_o(t)$ and desired output spike train $s_d(t)$ at time t. It can be represented as:

$$E(t) = \frac{1}{2}\left[f_{s_o}(t) - f_{s_d}(t)\right]^2 \tag{11}$$

So, the total error of the network in the time interval Γ is $E = \int_\Gamma E(t)dt$.

2.2.1. Learning Rule of Synaptic Weights

According to the gradient descent rule, the change of synaptic weight Δw_i from the presynaptic neuron i to the postsynaptic neuron is computed as follows:

$$\Delta w_i = -\eta \nabla E_w \tag{12}$$

where η is the learning rate of synaptic weights and ∇E_w is the gradient of the spike train error function E for the synaptic weight w_i. The gradient can be expressed as the integration of the derivative of the instantaneous error $E(t)$ with respect to synaptic weight w_i in the time interval Γ:

$$\nabla E_w = \int_\Gamma \frac{\partial E(t)}{\partial w_i} dt \tag{13}$$

Using the chain rule, the derivative of the error function $E(t)$ at time t to synaptic weight w_i can be represented as the product of two partial derivative terms:

$$\frac{\partial E(t)}{\partial w_i} = \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial w_i} \tag{14}$$

According to Equation (11), the first partial derivative term of the right-hand part of Equation (14) is computed as:

$$\frac{\partial E(t)}{\partial f_{s_o}(t)} = \frac{\partial \left[\frac{1}{2} \left[f_{s_o}(t) - f_{s_d}(t) \right]^2 \right]}{\partial f_{s_o}(t)} = f_{s_o}(t) - f_{s_d}(t) \tag{15}$$

According to Equation (10), the second partial derivative term of the right-hand part of Equation (14) is computed as:

$$\frac{\partial f_{s_o}(t)}{\partial w_i} = \frac{\partial \left[\sum_{i=1}^{N_I} w_i f_{s_i}(t - d_i) \right]}{\partial w_i} = f_{s_i}(t - d_i) \tag{16}$$

Therefore, the gradient ∇E_w in Equation (13) can be computed as follows according to Equations (15 and 16):

$$\nabla E_w = \int_\Gamma \left[f_{s_o}(t) - f_{s_d}(t) \right] f_{s_i}(t - d_i) dt \tag{17}$$

On the basis of the deduction process discussed above, a supervised learning rule of synaptic weights based on spike train kernels for spiking neurons with synaptic delays is given. The learning rule of the synaptic weights is expressed as follows:

$$\Delta w_i = -\eta \nabla E_w = \eta \int_\Gamma \left[f_{s_d}(t) - f_{s_o}(t) \right] f_{s_i}(t - d_i) dt \tag{18}$$

According to Equations (7–9), the synaptic weights learning can be further rewritten as:

$$\Delta w_i = \eta \left[\sum_{g=1}^{N_d} \sum_{f=1}^{N_i} \kappa \left(t_d^g - t_i^f - d_i \right) - \sum_{h=1}^{N_o} \sum_{f=1}^{N_i} \kappa \left(t_o^h - t_i^f - d_i \right) \right] \tag{19}$$

The learning rate η has a great influence on the convergence speed of the learning process, which can directly affect the training time and the training accuracy. Here we define an adaptive adjustment method of learning rate according to the firing rate of actual output spike train of neurons. Firstly, a scaling factor β is defined according to the different firing rates of the spike train. It is assumed that the firing rate of the spike train of neurons is r, and the referenced firing rate range is $[r_{min}, r_{max}]$. When $r \in [r_{min}, r_{max}]$, the scaling factor is $\beta = 1$; otherwise, the expression of β is:

$$\beta = \begin{cases} \frac{r_{min} - r}{r_{max} - r_{min}} & , r < r_{min} \\ \frac{r - r_{max}}{r_{max} - r_{min}} & , r > r_{max} \end{cases} \tag{20}$$

The learning rate in the referenced firing rate range is called the referenced learning rate η^*, and its value is the best learning rate for a given firing rate range. According to the scaling factor β and the referenced learning rate η^* in the firing rate range, the adaptive adjustment method of learning rate is:

$$\eta = \begin{cases} (1+\beta)\eta^* & , r < r_{min} \\ \eta^* & , r_{min} \le r \le r_{max} \\ \eta^*/(1+\beta) & , r > r_{max} \end{cases} \tag{21}$$

2.2.2. Learning Rule of Synaptic Delays

Here we derive the learning rule of synaptic delays with the similar derivation of synaptic weights. The synaptic delay change Δd_i from the presynaptic neuron i to the postsynaptic neuron is computed as follow:

$$\Delta d_i = -\alpha \nabla E_d \tag{22}$$

where α is the learning rate of synaptic delays and ∇E_d is the gradient of the spike train error function E for the synaptic delay d_i. The gradient can be expressed as the integration of the derivative of the instantaneous error $E(t)$ with respect to synaptic delay d_i in the time interval Γ:

$$\nabla E_d = \int_\Gamma \frac{\partial E(t)}{\partial d_i} dt \tag{23}$$

Using the chain rule, the derivative of the error function $E(t)$ to synaptic delay d_i at time t can be calculated as the product of two partial derivative terms:

$$\frac{\partial E(t)}{\partial d_i} = \frac{\partial E(t)}{\partial f_{s_o}(t)} \frac{\partial f_{s_o}(t)}{\partial d_i} \tag{24}$$

According to Equations (7 and 10), the second partial derivative term of the right-hand part of Equation (24) is computed as:

$$\begin{aligned} \frac{\partial f_{s_o}(t)}{\partial d_i} &= \frac{\partial \left[\sum_{i=1}^{N_I} w_i f_{s_i}(t - d_i) \right]}{\partial d_i} \\ &= \frac{\partial \left[\sum_{i=1}^{N_I} w_i \sum_{f=1}^{N_i} \kappa(t - t_i^f - d_i) \right]}{\partial d_i} \\ &= w_i \frac{\partial \left[\sum_{f=1}^{N_i} \kappa(t - t_i^f - d_i) \right]}{\partial d_i} \end{aligned} \tag{25}$$

For simplicity, here we choose the Laplacian kernel function to convert spike trains. It is defined as:

$$\kappa(s) = \exp\left(-\frac{|s|}{\tau}\right) \tag{26}$$

where τ is the scale parameter of the Laplacian kernel function. So the partial derivative term of the right-hand part of Equation (25) is computed as:

$$\begin{aligned}\frac{\partial\left[\sum_{f=1}^{N_i}\kappa(t-t_i^f-d_i)\right]}{\partial d_i} &= \frac{\partial\left[\sum_{f=1}^{N_i}\exp\left(-\frac{|t-t_i^f-d_i|}{\tau}\right)\right]}{\partial d_i}\\ &= \frac{1}{\tau}\sum_{f=1}^{N_i}\exp\left(-\frac{|t-t_i^f-d_i|}{\tau}\right)\\ &= \frac{1}{\tau}f_{s_i}(t-d_i)\end{aligned} \tag{27}$$

Therefore, on the basis of Equations (15), (25), and (27), the derivative $\partial E(t)/\partial d_i$ in Equation (23) can be further rewritten as:

$$\frac{\partial E(t)}{\partial d_i} = \frac{1}{\tau}w_i\left[f_{s_o}(t)-f_{s_d}(t)\right]f_{s_i}(t-d_i) \tag{28}$$

According to the deduction process discussed above, a supervised learning rule of synaptic delays based on spike train kernels for spiking neurons with Laplacian kernel is given. The learning rule of the synaptic delays is expressed as follows:

$$\Delta d_i = -\alpha\nabla E_d = \alpha\frac{1}{\tau}w_i\int_\Gamma\left[f_{s_d}(t)-f_{s_o}(t)\right]f_{s_i}(t-d_i)dt \tag{29}$$

According to Equations (7–9), the learning rule of synaptic delays can be further rewritten as:

$$\Delta d_i = \alpha\frac{1}{\tau}w_i\left[\sum_{g=1}^{N_d}\sum_{f=1}^{N_i}\kappa\left(t_d^g-t_i^f-d_i\right)-\sum_{h=1}^{N_o}\sum_{f=1}^{N_i}\kappa\left(t_o^h-t_i^f-d_i\right)\right] \tag{30}$$

2.3. Supervised Learning Algorithm for Spiking Neurons

Algorithm 1 represents the training process of spike train learning using our proposed supervised learning rule. In the beginning, we initialize all parameters of SNNs, mainly including the spiking neuron model and its parameters, the input and desired output spike trains, the synaptic weights and delays. Secondly, we calculate the actual output spike train of the output neuron according to the input spike trains and the spiking neuron model and then calculate the spike train error of the output neuron according to the actual and desired output spike train. Finally, we adjust all synaptic weights and delays according to our proposed learning rules of synaptic weights and delays. This process is called a learning epoch. Repeating the training process until the network error $E = 0$ or the upper limit of learning epochs is exceeded, the training process is ended.

Algorithm 1: supervised learning algorithm for spiking neurons.

```
1:  set up SNN
2:  initialize synaptic weights $w_i$ and delays $d_i$
3:  initialize input spike trains $s_i(t-d_i)$ and desired
     output spike trains $s_d(t)$
4:  calculate $f_{s_i}(t-d_i)$ according to $s_i(t-d_i)$
5:  calculate $f_{s_d}(t)$ according to $s_d(t)$
6:  repeat
7:      for all input neurons do
8:          input $s_i(t-d_i)$ into SNN
9:      end for
10:     for all output neurons do
11:         calculate output spike trains $s_o(t)$
12:         calculate $f_{s_o}(t)$ according to $s_o(t)$
13:         calculate network error $E$
14:     end for
15:     for all synapses do
16:         calculate learning rate of synaptic weights
            $\eta$
17:         calculate $\Delta w_i$
18:         $w_i \leftarrow w_i + \Delta w_i$
19:         calculate $\Delta d_i$
20:         $d_i \leftarrow d_i + \Delta d_i$
21:     end for
22: until network error $E = 0$ OR upper limit of
    learning epochs is exceeded
```

3. RESULTS

In this section, a series of spike train learning experiments and an image classification task are presented to demonstrate the learning capabilities of our proposed learning algorithm. At first, we analyze the learning process of our proposed algorithm. Then, we analyze the effects of the parameters of synaptic delays on learning performance, such as the learning rate of synaptic delays, the maximum allowed synaptic delays and the upper limit of learning epochs. In addition, we also analyze the effects of the parameters of network simulation on learning performance, such as the number of synaptic inputs, the firing rate of spike trains and the length of spike trains, and compare with the network with static synaptic delays on learning performance. Finally, we use the proposed delay learning algorithm to solve an image classification problem and compare with some other supervised learning algorithms for spiking neurons.

3.1. Parameter Settings and Learning Evaluation

Our experiments run on Java 1.7 on a quad-core system with 4-GB RAM in a Windows 10 environment. We use the clock-driven simulation strategy with time-step $dt = 0.1$ms to implement the spike train learning tasks. All reference parameters are shown in **Table 1**. Initially, the synaptic weights and the synaptic delays are generated as the uniform distribution in the interval $[w_{min}, w_{max}]$ and $[d_{min}, d_{max}]$, respectively. Every input spike train and desired

TABLE 1 | Reference parameters in the simulation.

Parameters		Identifiers	Value
	Number of input neurons	N_I	500
	Number of output neurons	N_O	1
Network simulation	Firing rate of input spike trains	r_{in}	20 Hz
	Firing rate of desired output spike trains	r_{out}	50 Hz
	Length of spike trains	Γ	200 ms
	Time constant of postsynaptic potential	τ	2 ms
SRM neuron model	Time constant of refractory period	τ_R	50 ms
	Spike firing threshold	θ	1
	Length of the absolute refractory period	t_R	1 ms
	Minimum synaptic weights	w_{min}	0
Synaptic weights	Maximum synaptic weights	w_{max}	0.5
	Referenced learning rate of synaptic weights	η^*	0.005
	Minimum synaptic delays	d_{min}	0 ms
Synaptic delays	Maximum synaptic delays	d_{max}	15 ms
	Learning rate of synaptic delays	α	3

output spike train is generated randomly by a homogeneous Poisson process within the time interval of Γ with firing rate r_{in} and r_{out}, respectively. Except for the learning process of spike trains demonstrated in section 3.2.1 and the image classification problem presented in section 3.3, the all simulation results are averaged over 100 trials, and on each testing trial, the learning algorithm is applied for a maximum of 500 learning epochs or until the network error $E = 0$. In the training process, the learning rate of synaptic weights is adjusted adaptively. The spiking neurons are described by the short-term memory SRM. The Laplacian kernel function $\kappa(s) = \exp(-|s|/\tau)$ with parameter $\tau = 10$ is used in all simulations.

To quantitatively evaluate the learning performance, we use the spike train kernels to define a measure C to express the distance between the desired output spike train $s_d(t)$ and the actual output spike train $s_o(t)$, which is equivalent to the correlation-based metric C (Schreiber et al., 2003). The metric is calculated after each learning epoch according to:

$$C = \frac{\langle f_{s_d}(t), f_{s_o}(t)\rangle}{\|f_{s_d}(t)\| \, \|f_{s_o}(t)\|} \tag{31}$$

where $\langle f_{s_d}(t), f_{s_o}(t)\rangle$ is the inner product of $f_{s_d}(t)$ and $f_{s_o}(t)$. $\|f_{s_d}(t)\| = \sqrt{\langle f_{s_d}(t), f_{s_d}(t)\rangle}$ and $\|f_{s_o}(t)\| = \sqrt{\langle f_{s_o}(t), f_{s_o}(t)\rangle}$ are the Euclidean norms of convolved continuous functions corresponding to spike trains $s_d(t)$ and $s_o(t)$, respectively. In order to keep in line with the measure described in Schreiber et al. (2003), here we use the Gaussian filter function to convert the spike trains. Measure $C = 1$ for identical spike trains and decreases toward 0 for loosely correlated spike trains.

3.2. Learning Sequences of Spikes

3.2.1. Analysis of the Learning Process

Figure 3 demonstrates the spike train learning process of one trial using the proposed synaptic delay learning rule to reproduce the desired output spatio-temporal spike pattern. **Figure 3A** shows the complete learning process in the time interval Γ, which includes the desired output spike train, the initial output spike train before learning and the actual output spike trains during the learning process. It can be seen that the actual output spike trains are closer to the desired output spike train during the learning process. The evolution of learning accuracy with measure C during the learning process is presented in **Figure 3B**. During the learning process, especially in the early stage, dithering occurs easily. However, the learning accuracy C increases gradually. After 30 learning epochs, the learning accuracy C reached 1.0. The synaptic delays before and after learning are shown in **Figures 3C,D**, respectively. These learning results show that the spiking neuron can successfully learn the desired output spike train using the proposed synaptic delay learning algorithm.

3.2.2. Parametric Analysis of Synaptic Delays

Here we test our proposed delay learning algorithm with the different learning rates of synaptic delays α, the maximum allowed synaptic delays d_{max} and the upper limit of learning epochs. **Figure 4** shows the learning results of delay learning algorithm with the different learning rates of synaptic delays α. The α takes 0.05, 0.5, 1.0, 2.0, 3.0, 5.0, 8.0, 10.0 in total of eight values. The learning accuracy with measure C after 500 learning epochs is shown in **Figure 4A**. It can be seen that the measure C increases slightly when α increases gradually. When $\alpha = 3.0$, the learning accuracy is $C = 0.9874$. When α increases further, the measure C decreases slightly, in addition, the standard deviation increased. When $\alpha = 8.0$, the learning accuracy is $C = 0.9664$. **Figure 4B** shows the learning epochs when the learning accuracy C reaches the maximum value. From **Figure 4B** we can see that when α increases gradually, the learning epochs do not change too much. When $\alpha = 3.0$, the mean learning epoch is 276.07. When $\alpha = 8.0$, the mean learning epoch is 249.14. This simulation indicates that the proposed delay learning algorithm can well learn with the different learning rates of synaptic delays in a large range. In the rest of the simulations, the learning rate of synaptic delays is $\alpha = 3.0$.

Neuroscience experiments give evidence to the variability of synaptic delay values, from 0.1 to 44 ms (Swadlow, 1992; Toyoizumi et al., 2005; Paugam-Moisy et al., 2008). This simulation tests the proposed delay learning algorithm with the different maximum allowed synaptic delays d_{max}, the learning results are shown in **Figure 5**. d_{max} increases from 5 to 30 ms with an interval of 5 ms. **Figure 5A** shows the learning accuracy with measure C after 500 learning epochs. From **Figure 5A** we can see that the delay learning algorithm can learn with high learning accuracy. The learning accuracy C basically remains the same when d_{max} less than 20 ms. When d_{max} increases further, the learning accuracy decreases, in addition, the standard deviation is increasing. For example, when $d_{max} = 10$ ms, the learning accuracy is $C = 0.9821$. When $d_{max} = 25$ ms, the learning accuracy is $C = 0.9629$. **Figure 5B** shows the learning epochs

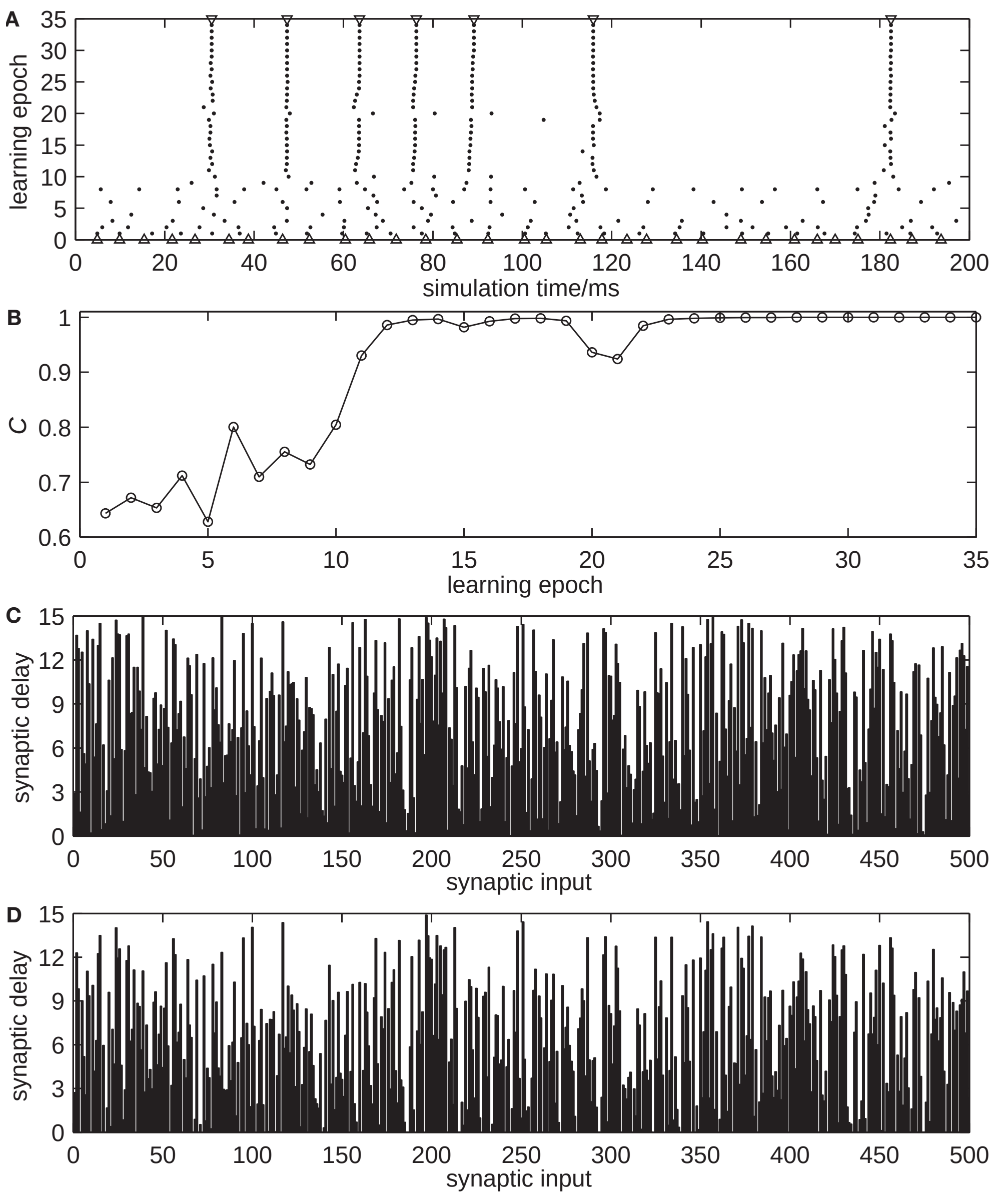


FIGURE 3 | The spike train learning process of proposed synaptic delay learning algorithm. **(A)** The complete learning process. △, the initial actual output spike train before learning; ▽, desired output spike train; •, actual output spike trains during the learning process. **(B)** The evolution of learning accuracy with measure *C*. **(C)** The synaptic delays before learning. **(D)** The synaptic delays after learning.

when the learning accuracy *C* reaches the maximum value. It can be seen that the learning epochs do not change too much when d_{max} increases gradually. For example, when $d_{max} = 10$ ms, the mean learning epoch is 274.06. When $d_{max} = 25$ ms, the mean learning epoch is 242.68. This simulation indicates that the proposed delay learning algorithm can learn from different maximum synaptic delays d_{max} in a large range. It is robust for various synaptic delays. In the rest of the simulations, the maximum synaptic delays is $d_{max} = 15$ ms.

The upper limit of learning epochs is a relatively important evaluation factor for supervised learning. If the upper limit of learning epochs is too small, the network cannot be fully trained, which will lead to the problem that the model cannot solve problems well. Conversely, if the upper limit of learning epochs is too large, it will take too much time to train the network. In this simulation, we test the proposed delay learning algorithm with the different upper limit of learning epochs, the learning results are shown in **Figure 6**. The upper limit of learning epochs increases from 100 to $1,000$ with an interval of 100, while the other settings remain the same. **Figure 6A** shows the learning accuracy with measure *C*. It can be seen that in the beginning, the learning accuracy *C* increases when the upper limit of learning

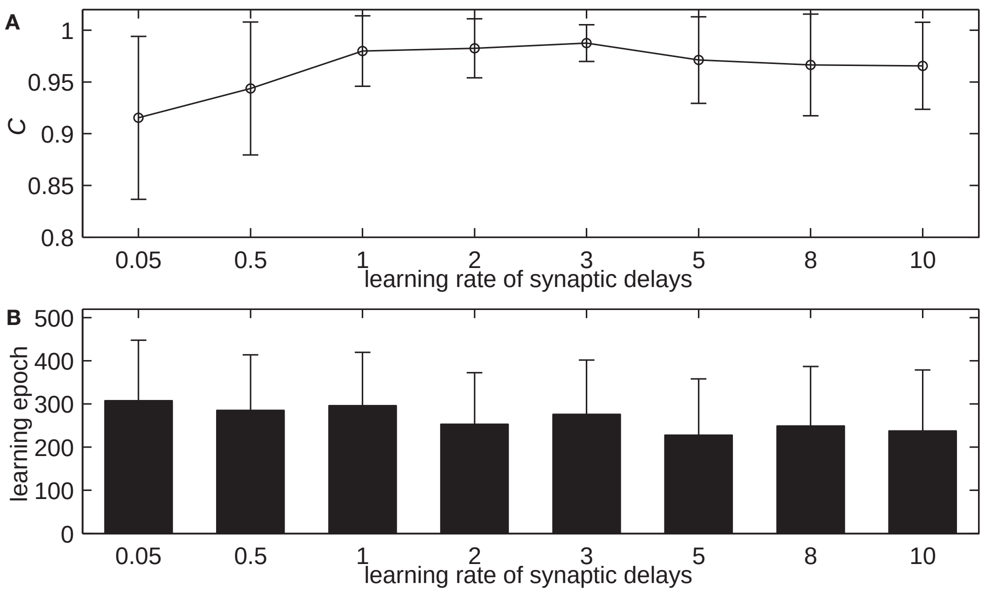


FIGURE 4 | The learning results with the different learning rates of synaptic delays α after 500 learning epochs. **(A)** The learning accuracy C. **(B)** The learning epochs when the learning accuracy C reaches the maximum value.

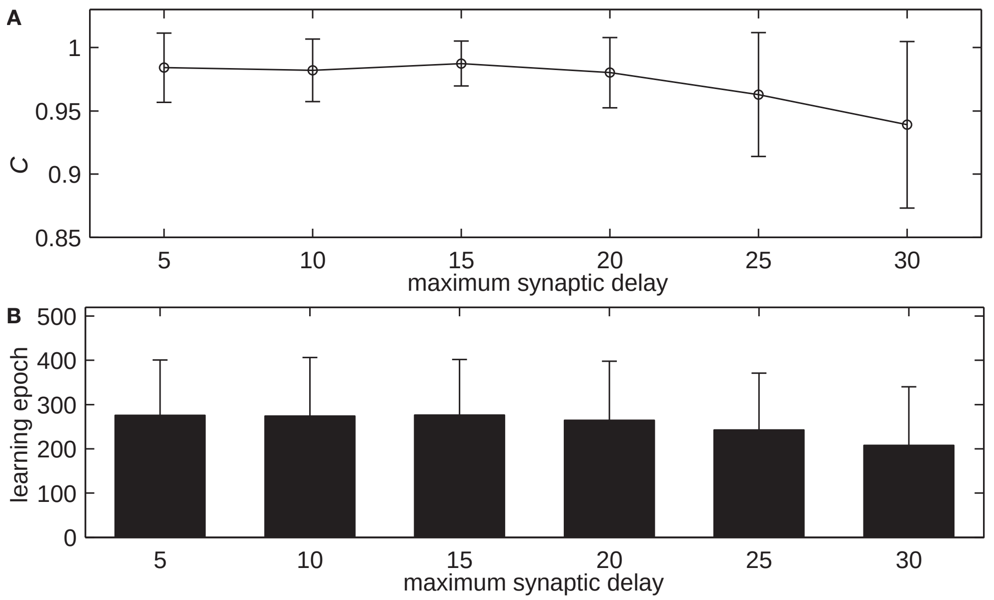


FIGURE 5 | The learning results with the different maximum allowed synaptic delays d_{max} after 500 learning epochs. **(A)** The learning accuracy C. **(B)** The learning epochs when the learning accuracy C reaches the maximum value.

epochs increases gradually. When the upper limit of learning epochs increases further, the learning accuracy C does not change too much. For example, when the upper limit of learning epochs is 400, the learning accuracy is $C = 0.9849$. When the upper limit of learning epochs is 800, the learning accuracy is $C = 0.9850$. **Figure 6B** shows the learning epochs when the learning accuracy C reaches the maximum value. From **Figure 6B** we can see that when the upper limit of learning epochs increases gradually, the actual learning epochs increase. When the upper limit of learning epochs is 600, the mean learning epoch is 315.78. When the upper limit of learning epochs increases further, the actual learning epochs do not change too much, but the standard deviation is increasing. When the upper limit of learning epochs is 900, the mean learning epoch is 330.98. This simulation indicates

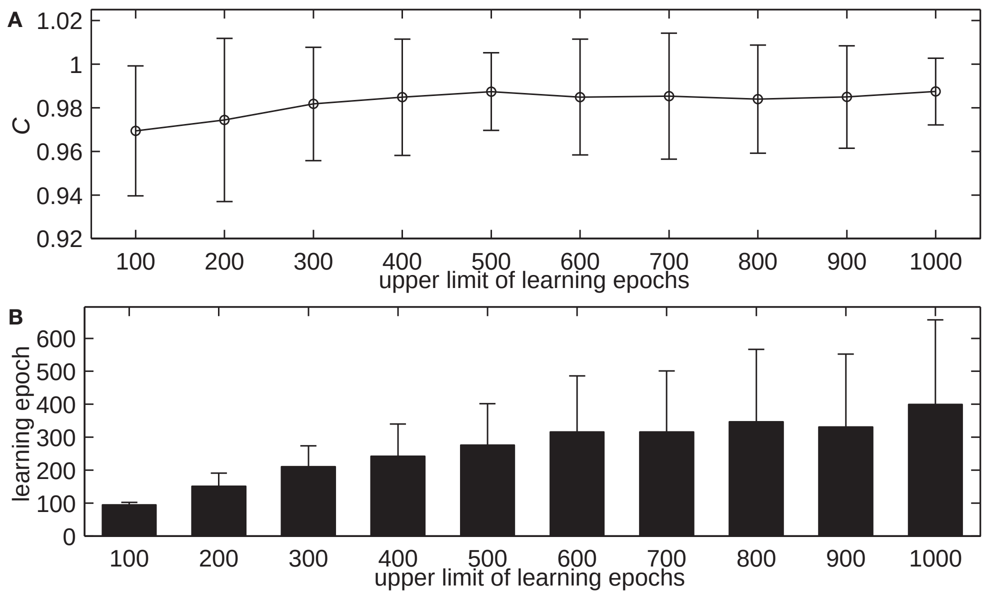


FIGURE 6 | The learning results with the different upper limit of learning epochs. **(A)** The learning accuracy *C*. **(B)** The learning epochs when the learning accuracy *C* reaches the maximum value.

that the proposed delay learning algorithm can learn with high learning accuracy, and increasing the upper limit of learning epochs cannot significantly improve learning accuracy. In the rest of the simulations, the upper limit of learning epochs is 500.

3.2.3. Comparative Analysis With Static Synaptic Delays

In this section, we analyze the parameters of network simulation that may influence the learning performance of delay learning algorithm and compare with the network with static synaptic delays on learning performance. The first simulation demonstrates the learning ability of our method with the different numbers of synaptic input N_I. The learning results are shown in **Figure 7**. The N_I increases from 100 to $1,000$ with an interval of 100, while the other settings remain the same. **Figure 7A** shows the learning accuracy after 500 learning epochs. It can be seen that both the network with dynamic delays and static delays can learn with high accuracy, but the learning accuracy of the network with dynamic delays is higher. The learning accuracy of both two methods increases when N_I increases gradually. For example, the measure $C = 0.9709$ for the network with dynamic delays and $C = 0.9189$ for the network with static delays when $N_I = 400$. When $N_I = 900$, the measure $C = 0.9941$ for the network with dynamic delays and $C = 0.9516$ for the network with static delays. **Figure 7B** shows the learning epochs when the measure C reaches the maximum value. From **Figure 7B** we can see that when N_I increases gradually, the learning epochs of both the network with dynamic delays and static delays are increased slightly, but the learning epochs of the network with dynamic delays are less than that of the network with static delays. When $N_I = 400$, the mean learning epoch is 266.83 for the network with dynamic delays and 338.89 for the network with static delays. When $N_I = 900$, the mean learning epoch is 314.95 for the network with dynamic delays and 368.82 for the network with static delays.

The second simulation demonstrates the learning ability of our proposed algorithm with the different firing rates of input and desired output spike trains. The learning results are shown in **Figure 8**. The firing rate of spike trains increases from 20 to 200 Hz with an interval of 20 Hz and the firing rate of input spike trains equals to that of desired output spike trains, while the other settings remain the same. **Figure 8A** shows the learning accuracy with measure C after 500 learning epochs. From **Figure 8A** we can see that when the firing rate of spike trains increases gradually, the learning accuracy of the network with dynamic delays decreases slightly, while the learning accuracy of the network with static delays decreases first, and then increases slightly, but the learning accuracy of the network with dynamic delays is higher than that of the network with static delays. For example, the measure $C = 0.9841$ for the network with dynamic delays and $C = 0.8588$ for the network with static delays when the firing rate of spike trains is 60 Hz. When the firing rate of spike trains is 140 Hz, the learning accuracy $C = 0.9504$ for the network with dynamic delays and $C = 0.8801$ for the network with static delays. **Figure 8B** shows the learning epochs when the learning accuracy C reaches the maximum value. It can be seen that the learning epochs of the network with dynamic delays are less than that of the network with static delays in the most case. When the firing rate of spike trains is 140 Hz, the mean learning epoch for the network with dynamic delays is 246.98, and 368.82 for the network with static delays.

The third simulation demonstrates the learning ability of our proposed algorithm with the different lengths of spike trains. The learning results are shown in **Figure 9**. The length of spike trains

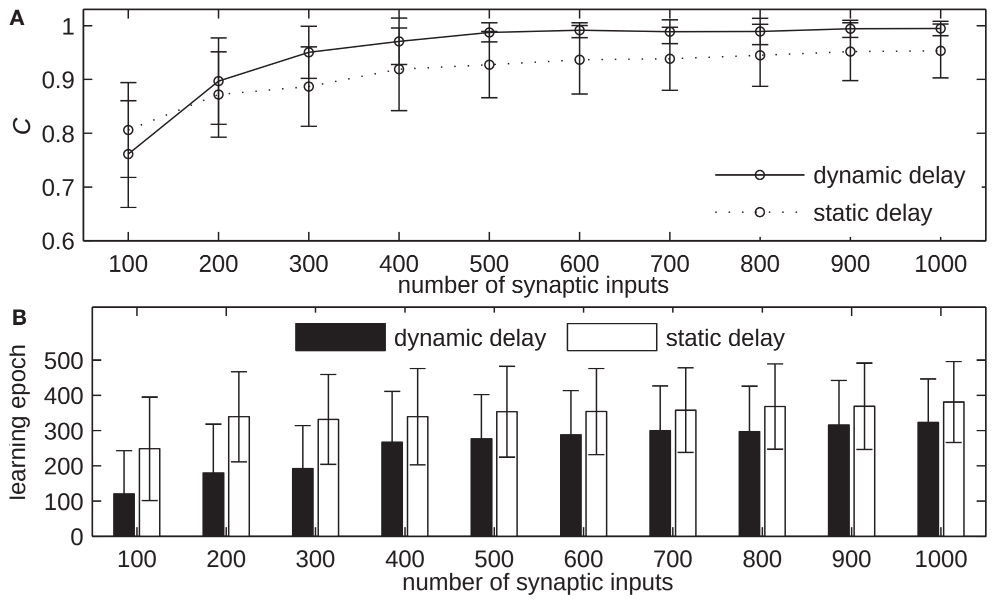


FIGURE 7 | The learning results with the different numbers of synaptic input N_I for the network with dynamic delays and static delays after 500 learning epochs. **(A)** The learning accuracy C. **(B)** The learning epochs when the learning accuracy C reaches the maximum value.

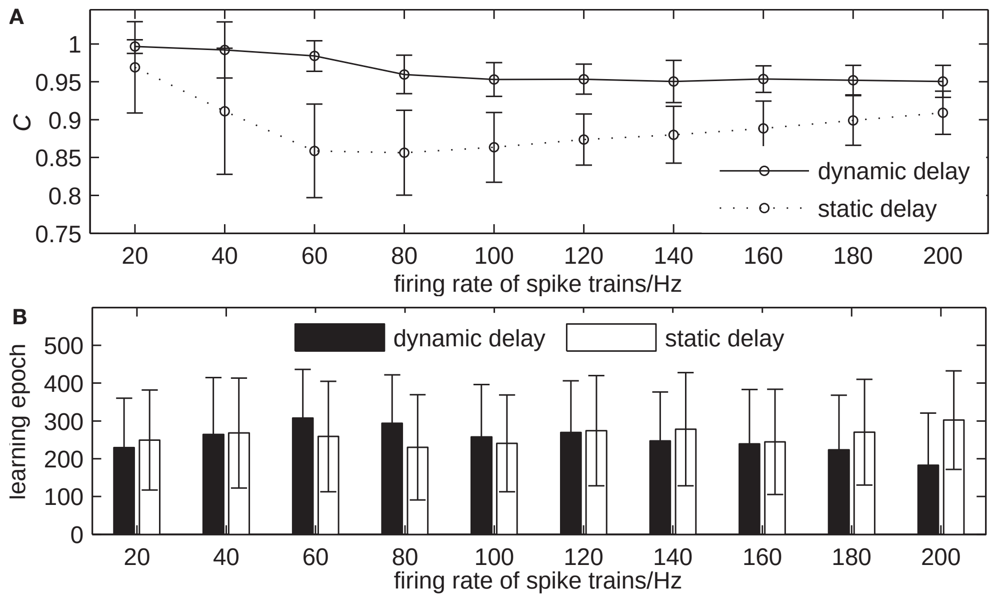


FIGURE 8 | The learning results with the different firing rates of spike trains for the network with dynamic delays and static delays after 500 learning epochs. **(A)** The learning accuracy C. **(B)** The learning epochs when the learning accuracy C reaches the maximum value.

increases from 100 to 1,000 ms with an interval of 100 ms, while the other settings remain the same. **Figure 9A** shows the learning accuracy C after 500 learning epochs. It can be seen that the learning accuracy of both the network with dynamic delays and static delays decreases when the length of spike trains increases gradually, but the learning accuracy of the network with dynamic delays is higher. For example, the learning accuracy $C = 0.9767$ for the network with dynamic delays and $C = 0.8743$ for the network with static delays when the length of spike trains is 300 ms. When the length of spike trains is 700 ms, the learning accuracy $C = 0.9461$ for the network with dynamic delays and $C = 0.7460$ for the network with static delays. **Figure 9B** shows the learning epochs when the learning accuracy C reaches the maximum value. It can be seen that the learning epochs of the

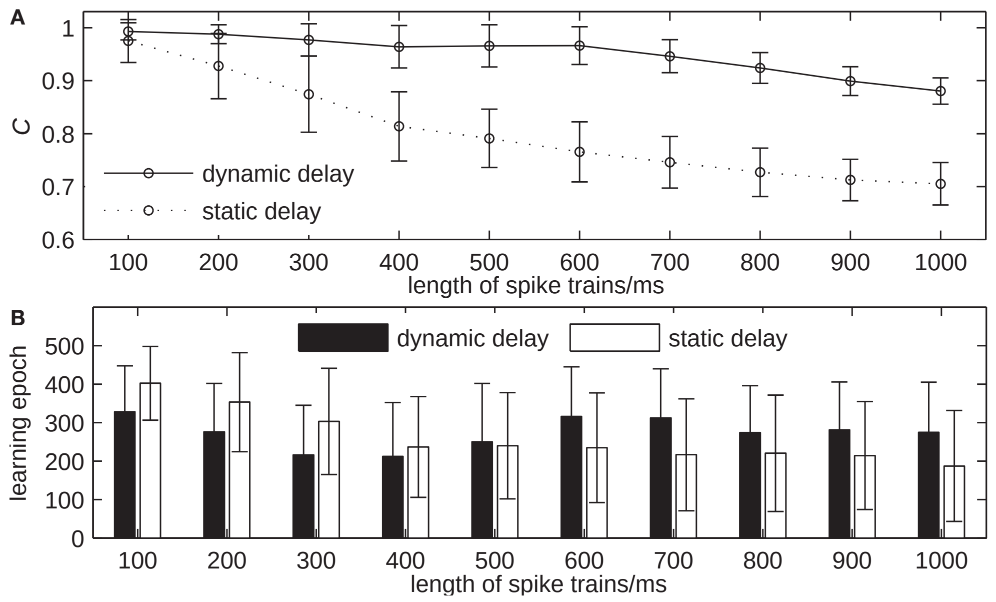


FIGURE 9 | The learning results with the different lengths of spike trains for the network with dynamic delays and static delays after 500 learning epochs. **(A)** The learning accuracy *C*. **(B)** The learning epochs when the learning accuracy *C* reaches the maximum value.

network with dynamic delays are less than that of the network with static delays when the length of spike trains is short. For example, when the length of spike trains is 300 ms, the mean learning epoch for the network with dynamic delays is 215.68, and 302.86 for the network with static delays.

3.3. Image Classification

3.3.1. Simulation Setup

Here we use the proposed delay learning algorithm to solve an image classification problem, and compare with some other supervised learning algorithms for spiking neurons. The general structure of the network for image classification is shown in **Figure 10**. It contains 2 functional parts: encoding and learning. In the encoding part, the latency-phase encoding method (Nadasdy, 2009) is used to transform the pixels of the image receptive field into precisely timed spike trains. In the learning part, each spike train corresponding to an input neuron is input into the spiking neural networks. The synaptic weights and delays are learned by the proposed delay learning algorithm. The spiking neural network outputs the target spike pattern for given images.

We choose the outdoor road images and the outdoor city street images from the LabelMe dataset (Russell et al., 2008) in the simulation. Each kind of images includes 20 samples, in a total of 40 samples. **Figure 11** shows some typical outdoor road images (top) and outdoor city street images (bottom). In our simulation, we choose 10 samples randomly from the outdoor road images and the outdoor city street images respectively (in total 20 samples, 50%) to constitute the training set, while the remaining 20 samples (50%) are constituted the testing set. The original images are converted into 256×256 gray images and then encoded into spike trains by the latency-phase encoding. In addition, we need to set the desired output spike trains of two kinds of images. The desired output spike train of the outdoor road images is set as $[20, 40, 60, 80]$ ms, while that of the outdoor city street images is set as $[40, 60, 80, 100]$ ms. The upper limit of learning epochs in the image classification is 50, and each result is averaged over 20 trials.

3.3.2. Learning With Different Sizes of Receptive Field

Table 2 shows the image classification accuracy on the testing set of the LabelMe dataset with different sizes of receptive field. The number of input neurons N_I equals the size of an image divided by the size of receptive field *RF*. The size of receptive field takes 2×2, 4×4, 8×8, 16×16, 32×32, and 64×64 in totals of six values. As seen from the table, with the increasing of *RF*, the testing accuracy of both the network with dynamic delays and static delays are firstly increased, and then decreased. In addition, the testing accuracy of the network with dynamic delays is higher than that of the network with static delays. When the size of the receptive field is 8×8, the testing accuracy of both the network with dynamic delays and static delays reached the highest 99.17 and 98.75%, respectively. The receptive field cannot be too large or too small. The appropriate size of the receptive field will obtain higher testing accuracy. The simulation results show that the proposed delay learning algorithm can be applied to image classification problem and achieve high classification accuracy.

3.3.3. Compare With Other Algorithms

The ReSuMe algorithm (Ponulak and Kasiński, 2010) has been used to solve the image classification problem (Hu et al., 2013), while the DL-ReSuMe algorithm (Taherkhani et al., 2015a) is a ReSuMe-based delay learning algorithm. In addition, SPAN (Mohemmed et al., 2012) and PSD (Yu et al., 2013) are two typical supervised learning algorithms for spiking neurons based

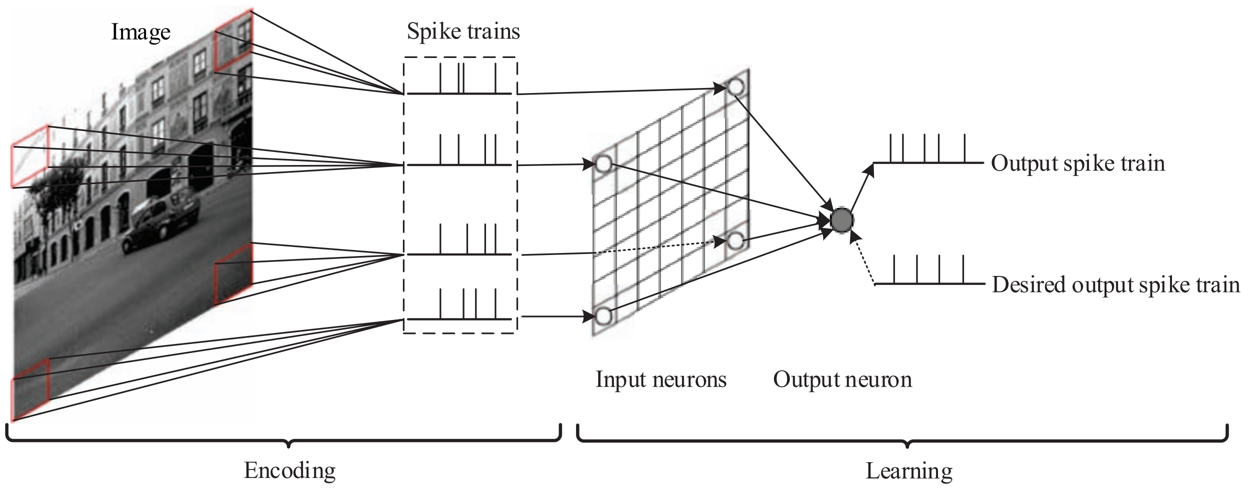


FIGURE 10 | Network structure for image classification.

FIGURE 11 | Some images from the LabelMe dataset.

on spike train convolution, which are similar to our proposed learning algorithm. Therefore, we use our proposed learning algorithm and DL-ReSuMe, ReSuMe, SPAN, PSD to solve the image classification problem, and further compare the image classification accuracy of these algorithms. The size of the receptive field is 8×8. The resulting image classification accuracy of these algorithms on the testing set is shown in **Figure 12**. The image classification accuracy of these algorithms on the testing set is 99.17% (dynamic delays), 98.75% (static delays), 98.74% (DL-ReSuMe), 97.56% (ReSuMe), 97.78% (SPAN), and 97.92% (PSD), respectively. It can be seen that all these algorithms can achieve high classification accuracy, but the accuracy of the network with dynamic delays is the highest.

4. DISCUSSION

In section 2.2.1, we introduced a supervised learning rule of synaptic weights based on spike train kernels for spiking neurons. The spike train is converted to a unique continuous function through a specific kernel function using the convolution. Then we construct the spike train error function through the convolved continuous functions corresponding to the actual output spike train and desired output spike train, and further deduce the supervised learning rule of synaptic weights by gradient descent method. The learning rule of synaptic weights is finally represented as the form of spike train kernels, which is similar to SPAN (Mohemmed et al., 2012) and PSD (Yu et al., 2013). It can be seen as a general framework of supervised learning algorithms for spiking neurons based on spike train convolution, in which different kernel functions can be used. The derivation of our proposed learning algorithm is independent of the spiking neuron model; it can be theoretically applied to any spiking neuron models. In the training process, the learning rate of synaptic weights is adjusted adaptively according to the firing rate of actual output spike train of neurons.

A new supervised learning rule of synaptic delays based on spike train kernels for spiking neurons is presented in section 2.2.2. The learning rule of synaptic delays is finally represented as the form of spike train kernels, which is similar to the learning rule of synaptic weights. For the sake of simplicity, we use the Laplacian kernel function in the derivation of learning rules. In fact, the general expression of the learning rule of synaptic delays is:

$$\Delta d_i = \alpha w_i \int_{\Gamma} \left\{ \left[f_{s_d}(t) - f_{s_o}(t) \right] \frac{\partial \left[\sum_{f=1}^{N_i} \kappa(t - t_i^f - d_i) \right]}{\partial d_i} \right\} dt \tag{32}$$

TABLE 2 | The image classification accuracy on the testing set with different sizes of receptive field.

RF	N_l	**Dynamic delays**	**Static delays**
2 × 2	16,384	90.36% ± 0.09	89.54% ± 0.07
4 × 4	4,096	92.68% ± 0.06	91.39% ± 0.07
8 × 8	1,024	99.17% ± 0.03	98.75% ± 0.03
16 × 16	256	97.46% ± 0.05	95.41% ± 0.05
32 × 32	64	91.40% ± 0.08	89.73% ± 0.08
64 × 64	16	90.80% ± 0.11	85.21% ± 0.13

In theory, as long as the kernel function $\kappa(t - t_i^f - d_i)$ is differentiable to d_i, such kernel functions can be used in the delay learning rule. If we choose different kernel functions, then the expression of the partial derivative in Equation (32) is different, and consequently, the expression of Δd_i is different.

There are some supervised delay learning algorithms for SNNs have been proposed in recent years. The first kind of supervised delay learning algorithms is ReSuMe-based delay learning algorithms (Taherkhani et al., 2015a,b, 2018; Guo et al., 2017). These algorithms merge the delay shift approach and ReSuMe-based weight adjustment (Ponulak and Kasiński, 2010) to enhance the learning performance of the original ReSuMe algorithm. Corresponding to the learning rules of synaptic weights, these algorithms can be regarded as supervised synaptic delay learning algorithms based on synaptic plasticity. The second kind of supervised delay learning algorithms is SpikeProp-based delay learning algorithms (Schrauwen and Van Campenhout, 2004; Matsuda, 2016; Shrestha and Song, 2016). These algorithms provide additional learning rule for the synaptic delays to improve the learning ability of the SpikeProp algorithm (Bohte et al., 2002). Similarly, these algorithms can be regarded as supervised synaptic delay learning algorithms based on gradient descent rule. There are also some other delay learning algorithms (Napp-Zinn et al., 1996; Wang et al., 2012; Hussain et al., 2014; Matsubara, 2017) have been proposed. Our proposed delay learning algorithm employs the spike train kernel to construct the error function, and then deduce the supervised learning rules of synaptic weights and delays. It can be seen as supervised synaptic delay learning algorithms based on spike train convolution. The kernel function is important for this kind of algorithm, in which different kernel functions can lead to different expressions of delay learning rule. It is an open question to consider which kernel function to choose in theory and practical application.

Analysis of the simulations in section 3 indicates that the proposed delay learning algorithm can obtain comparable learning results with different learning parameters. At first, the algorithm is applied to the learning sequences of spikes. The learning results show that the proposed delay learning algorithm can successfully learn the desired output spike train. Then

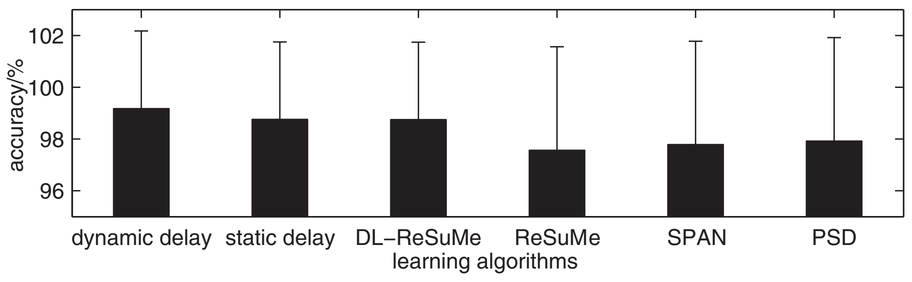


FIGURE 12 | The image classification accuracy of different algorithms.

TABLE 3 | Learning accuracy C of the delay learning algorithm.

	Static weights	Dynamic weights
Static delays	0.6123 ± 0.0862	0.9274 ± 0.0616
Dynamic delays	0.6528 ± 0.0739	0.9874 ± 0.0178

the parameters of synaptic delays are analyzed by simulation of spike train learning. The learning results show that the proposed delay learning algorithm can learn with the different learning rates of synaptic delays and the maximum allowed synaptic delays in a large range. The upper limit of learning epochs is also analyzed. The simulation results show that after 500 learning epochs, the proposed delay learning algorithm can obtain a relatively high learning accuracy. In addition, we analyze the factors that may influence the learning performance and compare with the network of static synaptic delays on learning performance. The simulation results show that the network with dynamic synaptic delays achieved higher learning accuracy and less learning epochs than that of the network with static synaptic delays. When the number of synaptic inputs increases, the learning accuracy of network with dynamic synaptic delays increases. When the firing rate of spike trains or the length of spike trains increases, the learning accuracy of network with dynamic synaptic delays decreases. Finally, we use the proposed delay learning algorithm to solve an image classification problem and archived higher classification accuracy in comparison of other similar supervised learning algorithms for spiking neurons.

The synaptic weight training is the dominant element of supervised learning for SNNs. However, delay training can improve the learning accuracy of SNNs. We tested the learning results of dynamic weights versus static weights under benchmark conditions (**Table 1**) over 100 trials. The corresponding learning accuracy C is shown in **Table 3**. When both the synaptic delays and weights are static, which means the random initial state of the SNNs, the learning accuracy is $C = 0.6123$. When the synaptic weights are static while the synaptic delays are dynamic, the learning accuracy is $C = 0.6528$. It shows that the dynamic delays can improve learning accuracy. When the synaptic weights are dynamic while the synaptic delays are static, the learning accuracy is $C = 0.9274$, which is significantly higher than that of the network with static weights. When both the synaptic delays and weights are dynamic, the learning accuracy $C = 0.9874$ is height. In summary, both the synaptic weights and delays have an impact on network training, but the impact of synaptic weights is greater. Delay training cannot replace weight training but can improve the learning accuracy of SNNs.

5. CONCLUSION

In this paper, we introduced a new supervised delay learning algorithm based on spike train kernels for spiking neurons. In this method, both the synaptic weights and the synaptic delays can be adjusted. We applied the proposed algorithm to a series of spike train learning experiments and an image classification problem to demonstrate the learning ability of spike train spatio-temporal pattern, and compared with the network with static synaptic delays on learning performance. Simulation results show that both the network with dynamic delays and static delays can successfully learn a random spike train and solve image classification problem, and the network with dynamic delays has higher learning accuracy and less learning epochs than that of the network with static delays.

Generally speaking, the more complex a neural network is, the more powerful its computing power is. The proposed supervised learning algorithm of synaptic delays in this paper can be applied only for a single layer SNNs, which limits the computing power of SNNs. We have proposed two supervised learning algorithms of synaptic weights for multi-layer feed-forward SNNs (Lin et al., 2017) and recurrent SNNs (Lin and Shi, 2018) based on inner products of spike trains. In the future work, we will extend the proposed delay learning algorithm to multi-layer feed-forward SNNs and recurrent SNNs to solve more complex and practical spatio-temporal pattern recognition problems.

DATA AVAILABILITY

Publicly available datasets were analyzed in this study. This data can be found here: http://labelme.csail.mit.edu/Release3.0/.

AUTHOR CONTRIBUTIONS

XW wrote the paper and performed the simulations. XL conceived the theory and designed the simulations. XD discussed about the results and analysis, and reviewed the manuscript. All authors helped with developing the concepts, conceiving the simulations, and writing the paper.

FUNDING

This work is supported by the National Natural Science Foundation of China under Grants No. 61762080 and No. 61662070, and the Program for Innovative Research Team in Northwest Normal University under Grant No. 6008-01602.

REFERENCES

Beyeler, M., Dutt, N. D., and Krichmar, J. L. (2013). Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule. *Neural Netw.* 48, 109–124. doi: 10.1016/j.neunet.2013.07.012

Bohte, S. M., Kok, J. N., and Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Boudkkazi, S., Fronzaroli-Molinieres, L., and Debanne, D. (2011). Presynaptic action potential waveform determines cortical synaptic latency. *J. Physiol.* 589, 1117–1131. doi: 10.1113/jphysiol.2010.199653

Carnell, A., and Richardson, D. (2005). "Linear algebra for times series of spikes," in *Proceedings of 2005 European Symposium on Artificial Neural Networks* (Bruges), 363–368.

Cash, S., and Yuste, R. (1999). Linear summation of excitatory inputs by CA1 pyramidal neurons. *Neuron* 22, 383–394. doi: 10.1016/S0896-6273(00)81098-3

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.

Ghosh-Dastidar, S., and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308. doi: 10.1142/S0129065709002002

Guo, L., Wang, Z., Cabrerizo, M., and Adjouadi, M. (2017). A cross-correlated delay shift supervised learning method for spiking neurons with application to interictal spike detection in epilepsy. *Int. J. Neural Syst.* 27:1750002. doi: 10.1142/S0129065717500022

Gütig, R. (2014). To spike, or when to spike? *Curr. Opin. Neurobiol.* 25, 134–139. doi: 10.1016/j.conb.2014.01.004

Hu, J., Tang, H., Tan, K. C., Li, H., and Shi, L. (2013). A spike-timing-based integrated model for pattern recognition. *Neural Comput.* 25, 450–472. doi: 10.1162/NECO_a_00395

Hussain, S., Basu, A., Wang, R. M., and Hamilton, T. J. (2014). Delay learning architectures for memory and classification. *Neurocomputing* 138, 14–26. doi: 10.1016/j.neucom.2013.09.052

Lin, J.-W., and Faber, D. S. (2002). Modulation of synaptic delay during synaptic plasticity. *Trends Neurosci.* 25, 449–455. doi: 10.1016/S0166-2236(02)02212-9

Lin, X., Chen, G., Wang, X., and Ma, H. (2016). "An improved supervised learning algorithm using triplet-based spike-timing-dependent plasticity," in *International Conference on Intelligent Computing* (Lanzhou: Springer), 44–53.

Lin, X., Li, Q., and Li, D. (2018). "Supervised learning algorithm for multi-spike liquid state machines," in *International Conference on Intelligent Computing* (Wuhan: Springer), 243–253.

Lin, X., Ning, Z., and Wang, X. (2015a). "An online supervised learning algorithm based on nonlinear spike train kernels," in *International Conference on Intelligent Computing* (Fuzhou: Springer), 106–115.

Lin, X., and Shi, G. (2018). "A supervised multi-spike learning algorithm for recurrent spiking neural networks," in *International Conference on Artificial Neural Networks* (Rhodes: Springer), 222–234.

Lin, X., Wang, X., and Hao, Z. (2017). Supervised learning in multilayer spiking neural networks with inner products of spike trains. *Neurocomputing* 237, 59–70. doi: 10.1016/j.neucom.2016.08.087

Lin, X., Wang, X., Zhang, N., and Ma, H. (2015b). Supervised learning algorithms for spiking neural networks: a review. *Acta Electron. Sin.* 43, 577–586. doi: 10.3969/j.issn.0372-2112.2015.03.024

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Comput.* 8, 1–40. doi: 10.1162/neco.1996.8.1.1

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Matsubara, T. (2017). Conduction delay learning model for unsupervised and supervised classification of spatio-temporal spike patterns. *Front. Comput. Neurosci.* 11:104. doi: 10.3389/fncom.2017.00104

Matsuda, S. (2016). "BPSpike: a backpropagation learning for all parameters in spiking neural networks with multiple layers and multiple spikes," in *International Joint Conference on Neural Networks*(Vancouve, BC), 293–298.

Minneci, F., Kanichay, R. T., and Silver, R. A. (2012). Estimation of the time course of neurotransmitter release at central synapses from the first latency of postsynaptic currents. *J. Neurosci. Methods* 205, 49–64. doi: 10.1016/j.jneumeth.2011.12.015

Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. (2012). SPAN: spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* 22:1250012. doi: 10.1142/S0129065712500128

Nadasdy, Z. (2009). Information encoding and reconstruction from the phase of action potentials. *Front. Syst. Neurosci.* 3:6. doi: 10.3389/neuro.06.006.2009

Napp-Zinn, H., Jansen, M., and Eckmiller, R. (1996). Recognition and tracking of impulse patterns with delay adaptation in biology-inspired pulse processing neural net (BPN) hardware. *Biol. Cybern.* 74, 449–453. doi: 10.1007/BF00206711

Paiva, A. R. C., Park, I., and Príncipe, J. C. (2009). A reproducing kernel Hilbert space framework for spike train signal processing. *Neural Comput.* 21, 424–449. doi: 10.1162/neco.2008.09-07-614

Park, I. M., Seth, S., Paiva, A. R. C., Li, L., and Principe, J. C. (2013). Kernel methods on spike train space for neuroscience: a tutorial. *IEEE Signal Process. Mag.* 30, 149–160. doi: 10.1109/MSP.2013.2251072

Paugam-Moisy, H., Martinez, R., and Bengio, S. (2008). Delay learning and polychronization for reservoir computing. *Neurocomputing* 71, 1143–1158. doi: 10.1016/j.neucom.2007.12.027

Ponulak, F., and Kasiński, A. (2010). Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). LabelMe: a database and web-based tool for image annotation. *Int. J. Comput. Vis.* 77, 157–173. doi: 10.1007/s11263-007-0090-8

Schrauwen, B., and Van Campenhout, J. (2004). "Improving SpikeProp: enhancements to an error-backpropagation rule for spiking neural networks," in *Proceedings of the 15th ProRISC Workshop, Vol. 11* (Veldhoven), 301–305.

Schreiber, S., Fellous, J. M., Whitmer, D., Tiesinga, P., and Sejnowski, T. J. (2003). A new correlation-based measure of spike timing reliability. *Neurocomputing* 52, 925–931. doi: 10.1016/S0925-2312(02)00838-X

Schwartz, O., Pillow, J. W., Rust, N. C., and Simoncelli, E. P. (2006). Spike-triggered neural characterization. *J. Vis.* 6, 484–507. doi: 10.1167/6.4.13

Shrestha, S. B., and Song, Q. (2016). "Adaptive delay learning in SpikeProp based on delay convergence analysis," in *International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC), 277–284.

Swadlow, H. A. (1992). Monitoring the excitability of neocortical efferent neurons to direct activation by extracellular current pulses. *J. Neurophysiol.* 68, 605–619. doi: 10.1152/jn.1992.68.2.605

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015a). DL-ReSuMe: a delay learning-based remote supervised method for spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3137–3149. doi: 10.1109/TNNLS.2015.2404938

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015b). "Multi-DL-ReSuMe: multiple neurons delay learning remote supervised method," in *International Joint Conference on Neural Networks (IJCNN)* (Killarney), 1–7.

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2018). A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 99, 1–14. doi: 10.1109/TNNLS.2018.2797801

Toyoizumi, T., Pfister, J.-P., Aihara, K., and Gerstner, W. (2005). Generalized Bienenstock-Cooper-Munro rule for spiking neurons that maximizes information transmission. *Proc. Natl. Acad. Sci. U.S.A.* 102, 5239–5244. doi: 10.1073/pnas.0500495102

Walter, F., Röhrbein, F., and Knoll, A. (2016). Computation by time. *Neural Process. Lett.* 44, 103–124. doi: 10.1007/s11063-015-9478-6

Wang, R., Tapson, J., Hamilton, T. J., and van Schaik, A. (2012). "An aVLSI programmable axonal delay circuit with spike timing dependent delay adaptation," in *IEEE International Symposium on Circuits and Systems* (Seoul), 2413–2416.

Wang, X., Lin, X., Zhao, J., and Ma, H. (2016). "Supervised learning algorithm for spiking neurons based on nonlinear inner products of spike trains," in *International Conference on Intelligent Computing* (Lanzhou: Springer), 95–104.

Whalley, K. (2013). Neural coding: timing is key in the olfactory system. *Nat. Rev. Neurosci.* 14, 458–458. doi: 10.1038/nrn3532

Xu, B., Gong, Y., and Wang, B. (2013). Delay-induced firing behavior and transitions in adaptive neuronal networks with two types of synapses. *Sci. China Chem.* 56, 222–229. doi: 10.1007/s11426-012-4710-y

Xu, Y., Yang, J., and Zhong, S. (2017). An online supervised learning method based on gradient descent for spiking neurons. *Neural Netw.* 93, 7–20. doi: 10.1016/j.neunet.2017.04.010

Yu, Q., Tang, H., Tan, K. C., and Li, H. (2013). Precise-Spike-Driven synaptic plasticity: learning hetero-association of spatiotemporal spike patterns. *PLoS ONE* 8:e78318. doi: 10.1371/journal.pone.0078318

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 24 April 2019
doi: 10.3389/fnins.2019.00357

Memory-Efficient Synaptic Connectivity for Spike-Timing-Dependent Plasticity

Bruno U. Pedroni[1]*, *Siddharth Joshi*[2], *Stephen R. Deiss*[1], *Sadique Sheik*[3], *Georgios Detorakis*[4], *Somnath Paul*[5], *Charles Augustine*[5], *Emre O. Neftci*[4] and *Gert Cauwenberghs*[1]

[1] *Integrated Systems Neuroengineering Laboratory, Department of Bioengineering, University of California, San Diego, La Jolla, CA, United States,* [2] *Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, United States,* [3] *aiCTX, Zurich, Switzerland,* [4] *Department of Cognitive Sciences, University of California, Irvine, Irvine, CA, United States,* [5] *Intel Corporation - Circuit Research Lab, Hillsboro, OR, United States*

OPEN ACCESS

Edited by:
Chiara Bartolozzi,
Istituto Italiano di Tecnologia, Italy

Reviewed by:
James Courtney Knight,
University of Sussex, United Kingdom
Quansheng Ren,
Peking University, China
Alejandro Linares-Barranco,
Universidad de Sevilla, Spain

***Correspondence:**
Bruno U. Pedroni
bpedroni@eng.ucsd.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *22 November 2018*
Accepted: *28 March 2019*
Published: *24 April 2019*

Citation:
Pedroni BU, Joshi S, Deiss SR, Sheik S, Detorakis G, Paul S, Augustine C, Neftci EO and Cauwenberghs G (2019) Memory-Efficient Synaptic Connectivity for Spike-Timing-Dependent Plasticity. Front. Neurosci. 13:357. doi: 10.3389/fnins.2019.00357

Spike-Timing-Dependent Plasticity (STDP) is a bio-inspired local incremental weight update rule commonly used for online learning in spike-based neuromorphic systems. In STDP, the intensity of long-term potentiation and depression in synaptic efficacy (weight) between neurons is expressed as a function of the relative timing between pre- and post-synaptic action potentials (spikes), while the polarity of change is dependent on the order (causality) of the spikes. Online STDP weight updates for causal and acausal relative spike times are activated at the onset of post- and pre-synaptic spike events, respectively, implying access to synaptic connectivity both in forward (pre-to-post) and reverse (post-to-pre) directions. Here we study the impact of different arrangements of synaptic connectivity tables on weight storage and STDP updates for large-scale neuromorphic systems. We analyze the memory efficiency for varying degrees of density in synaptic connectivity, ranging from crossbar arrays for full connectivity to pointer-based lookup for sparse connectivity. The study includes comparison of storage and access costs and efficiencies for each memory arrangement, along with a trade-off analysis of the benefits of each data structure depending on application requirements and budget. Finally, we present an alternative formulation of STDP via a delayed causal update mechanism that permits efficient weight access, requiring no more than forward connectivity lookup. We show functional equivalence of the delayed causal updates to the original STDP formulation, with substantial savings in storage and access costs and efficiencies for networks with sparse synaptic connectivity as typically encountered in large-scale models in computational neuroscience.

Keywords: synaptic plasticity, neuromorphic computing, data structure, memory architecture, crossbar array

1. INTRODUCTION

Extensive research in the field of artificial neural networks (ANNs) in the past decade has given rise to diverse neuron functions, network topologies, and training techniques (Nair and Hinton, 2010; Krizhevsky et al., 2012; Goodfellow et al., 2014; Kingma and Ba, 2014; Ioffe and Szegedy, 2015), capable of solving complex cognitive tasks, such as image classification (Krizhevsky et al., 2012), sequence generation (Graves, 2013), speech recognition (Graves et al., 2013), and game playing

(Silver et al., 2016). However, the components of these algorithms are normally only loosely based on actual biological neural networks, particularly with respect to the non-local learning rules (e.g., the widely used backpropagation algorithm, Rumelhart et al., 1986) and the continuous activation functions (e.g., sigmoid unit and rectified linear unit). Spiking neural networks (SNNs), in contrast, incorporate multiple aspects of biological nervous systems into its components (Gerstner and Kistler, 2002), including biologically relevant neuron models, binary activation functions and communication, event-driven processing, and local learning rules (i.e., where all the information required for adjusting parameters between neurons is collocated with these neurons). The neuron models can range from simple single-variable differential equations (e.g., McCulloch-Pitts and integrate-and-fire), to complex systems with dynamics more homologous to real neurons (e.g., Hodgkin-Huxley). In SNNs, neurons communicate between each other via a binary event known as an action potential (or *spike*), which is elicited whenever a neuron variable (typically, the membrane potential) crosses a threshold value. Whenever a neuron produces an action potential, this spike event information is conveyed to its population of downstream post-synaptic neurons, resulting in an update of their respective internal variables based on the values of synaptic efficacy (or *weight*). Due to their binary nature, the time at which spikes occur is essential information when training SNNs.

The origins of hardware designed to emulate the biological nervous system, also known as neuromorphic systems Mead (1990), targeted design of neural properties at the device level, with natural focus on analog circuits (Maher et al., 1989; Andreou et al., 1995; Koch and Mathur, 1996). More recently, however, neuromorphic systems such as TrueNorth (Merolla et al., 2014), SpiNNaker (Furber et al., 2014), and Loihi (Davies et al., 2018) were designed with purely digital components, being capable of emulating large-scale SNNs with real-time dynamics in the millisecond timescale. Additionally, large digital systems have the advantage of being more readily verifiable in simulation and a software-hardware equivalence is typically possible. While ANNs operate in a sequential manner, where data propagates through the network one layer at a time, neuromorphic systems typically present multiple cores running in parallel at biological timescales, with synaptic memory local to each core. Systems with distributed processing and memory move away from the traditional von Neumann architecture, where memory is centralized and a high-frequency global clock is responsible for fast computation and memory access (Merolla et al., 2014).

Among the bio-inspired learning mechanisms, spike-timing-dependent plasticity (STDP) is perhaps the most widely considered form of induced synaptic modification (Markram et al., 1997). STDP originated from experimental data collected in cultures of dissociated rat hippocampal neurons, where scientists observed that a causal relationship between spike times of pre- and post-synaptic neurons could induce synaptic strengthening or weakening, and this change was correlated with the relative temporal difference of spikes (Bi and Poo, 1998). The experiments showed that long-term potentiation and long-term depression could both be induced in synapses depending on the order of spike occurrence, where a causal relationship (i.e., pre-synaptic neuron spikes before post-synaptic neuron) potentiated the synapse, while an acausal relationship (i.e., post-synaptic spikes before pre-synaptic) weakened the synapse. The authors then approximated the measured synaptic modification with a mathematical model. In the model, the STDP function (or *kernel*) defines the change of the weight as a function of the relative time between pre- and post-synaptic action potentials, and the duration of the causal (and acausal) influence of spikes is called the STDP learning window (Sjöström and Gerstner, 2010). An important aspect of STDP is that, though it is a local learning rule, weight updates occur at the onset of both pre- and post-synaptic spikes, requiring for the algorithm to be able to not only identify all neurons which the pre-synaptic neuron sends its spikes to, but also locate all the neurons which the post-synaptic neuron receives its spikes from. This is a fundamental property of STDP, and throughout our work we will refer to reading the neuron addresses and weights from pre-to-post connectivity as *forward access* and reading from post-to-pre connectivity as *reverse access*.

In traditional ANNs, the typical data structure used to represent the weights between neurons is a dense matrix, constituting a fully connected topology. However, more realistic and biologically relevant neural networks, such as small-world and locally connected random networks (Bassett and Bullmore, 2006; Bullmore and Sporns, 2009; Seeman et al., 2018), do not conform to this structured topology. In these cases, synaptic weight storage costs can benefit greatly using compressed representations. For physical realizations of the STDP learning rule, the arrangement used to organize the synaptic weights in memory has a direct impact on the ease of forward and reverse access. As we will later show, dense matrices typically have the advantage of natively facilitating both types of connectivity access. Conversely, compressed memory arrangements suffer greatly when trying to access in the reverse direction, making causal STDP weight updates in these structures computationally intensive. In this work, we discuss the complexity of storing and accessing synaptic weights in different types of data structures and their impact on implementations of the STDP algorithm, and propose a novel method of performing STDP using only single-direction connectivity access, consequently taking advantage of compressed structures.

Storage costs associated to synaptic weight memory arrangements have been previously studied (Moradi et al., 2013; Pedroni et al., 2016; Joshi et al., 2017; Kornijcuk et al., 2018). In Materials and Methods, we give an overview of four typical data structures used for representing synaptic weights, and analyze storage costs based on different network parameters (number of neurons and weight bit-length) and varying degrees of network connectivity density. We extend our analysis to verify the memory access cost and efficiency associated to each data structure, focusing particularly on the computational complexity and requirements for performing STDP. Inspired by our previous work (Pedroni et al., 2016), we propose a definite pre-synaptic-driven solution for obtaining a quantitatively equivalent algorithm to STDP. Previous attempts in approximating STDP using forward-only connectivity include (1) simplifying the STDP rule by equally updating all the synaptic weights based on recent spike activity (Bichler et al., 2012; Yousefzadeh et al., 2017), (2) using other variables (usually post-synaptic membrane potential) as a proxy for the post-synaptic spike times when computing causal updates (Brader et al., 2007; Davies et al., 2012; Lagorce et al., 2015; Sheik et al., 2016), and (3) delaying the weight updates (Jin et al., 2010; Davies et al., 2018). In the discussion, we compare our method to these, particularly with the third type, currently present in SpiNNaker and Loihi, and explain how our solution can produce exact STDP while previous methods rely on particular balanced firing rate conditions in the network or simply produce qualitative approximations to STDP. In Results, a network composed of 256 pre-synaptic and 256 post-synaptic neurons is simulated using our proposed method and compared against the original STDP learning rule, showing that our method produces the same post-synaptic membrane potentials, resulting in identical spiking activity and synaptic weights.

2. MATERIALS AND METHODS

2.1. Digital Neuromorphic Core

Neuromorphic systems emulate the biophysics of neural computation in correspondingly tailored electronic circuits (Mead, 1990). Whereas artificial neural networks are typically deployed as software applications in general purpose hardware, neuromorphic systems are normally developed accounting for the properties and limitations that a physical hardware implementation entails. These include biologically plausible neurons (i.e., spiking neurons) and learning rules, binary event communication (i.e., neurons communicating via spikes), limited and local synaptic memory, and parallel and distributed neuron processing (Mahowald, 1993; Liu and Delbruck, 2010; Indiveri et al., 2011; Park et al., 2017).

The current state-of-the-art digital neuromorphic processors, such as TrueNorth (Merolla et al., 2014) and Loihi (Davies et al., 2018), partition the network into *cores*, where typically the population of post-synaptic neurons in a core shares inputs from a common pool of pre-synaptic neurons. At a high level, the core comprises of a digital finite-state machine, with weights stored in digital memory elements (e.g., random access memory - RAM), and with the state of the neural and synaptic variables progressing in discrete time steps (Δt), representing the temporal precision of the system. **Figure 1A** illustrates an abstract digital neuromorphic core and its components. The core operates by processing incoming pre-synaptic spikes (irrespective of their origins) and updating the post-synaptic state variables (e.g., membrane potential) with the associated weight between the pre- and post-synaptic neurons. Once all pre-synaptic spikes have been processed, the post-synaptic neurons are evaluated. Any new post-synaptic spike is then routed to its destination (on another or the same core), where there it is treated as an incoming pre-synaptic spike and is buffered to be used in the next system time step.

For realizing STDP learning in digital neuromorphic systems, a core must locally store (or have access to) the following: pre-synaptic spike times, synaptic weights, and post-synaptic neurons and spike times. Collocating the synaptic weights with the post-synaptic neurons ensures that all the information required for local and distributed learning strategies can be accessed with minimum overhead (Joshi et al., 2017). Interestingly, since our proposed method operates in pre-synaptic spike-driven fashion, a core does not require storing the pre-synaptic spike times. In other words, the spike times only need to be stored at the origin of the spike (i.e., at the post-synaptic neuron).

Lastly, an important consideration throughout our work is that we analyze the storage and access efficiency of the different memory arrangements based on the data structure used for storing synaptic weights. For this, we abstract away the physical storage elements by considering that each position in memory contains only a single "packet" of information (of arbitrary length), and that only one position in memory can be accessed at a time (i.e., each read/write command targets one "packet" at a time). Though memory storage and access in dynamic RAMs (DRAMs), for example, is typically not performed on an arbitrary number of bits (i.e., usually each read/write command targets a few bytes at a time), and complete random access is less efficient than bursts of sequential addresses of data, understanding the efficiency of each memory arrangement would become too involved if we were to consider the intricacies of exact physical models. For simplicity, we consider that *storage costs* take into account only the total number of bits for storing the connectivity and weight tables, and that each read/write command accesses only one address of the table at a time. Thus, the computational complexity of locating neuron addresses and weights in the data structures, denoted as *access cost*, considers the number of variables which must be accessed until the desired information is located, and can perhaps serve as a proxy for indirectly evaluating latency and energy of the methods.

2.2. Spike-Timing-Dependent Plasticity (STDP)

Spike-Timing-Dependent Plasticity is a biologically inspired form of Hebbian learning which considers the relative spike time of pre- and post-synaptic neurons for updating the synaptic efficacy (or weight) (Caporale and Dan, 2008). Though STDP is believed to be a fundamental learning mechanism in the mammalian brain (Dan and Poo, 2004) and has been widely explored in computational neuroscience (Song and Abbott, 2001; Izhikevich, 2007; Sjöström and Gerstner, 2010), results obtained in machine learning applications (Nessler et al., 2009; Diehl and Cook, 2015; Yousefzadeh et al., 2017; Kheradpisheh et al., 2018) suggest it may also be an interesting solution in non-biological scenarios.

STDP operates by modifying synaptic weights at the onset of pre- and post-synaptic spikes. "Causal updates" occur when a pre-synaptic spike precedes a post-synaptic spike, resulting in an increase in synaptic efficacy (i.e., long-term potentiation). Conversely, when a pre-synaptic spike proceeds a post-synaptic spike, an "acausal update" occurs and the efficacy is reduced (i.e., long-term depression). **Figure 1B** identifies the causal and acausal regions of the STDP function. The strength in which these changes take place is dependent on the temporal difference

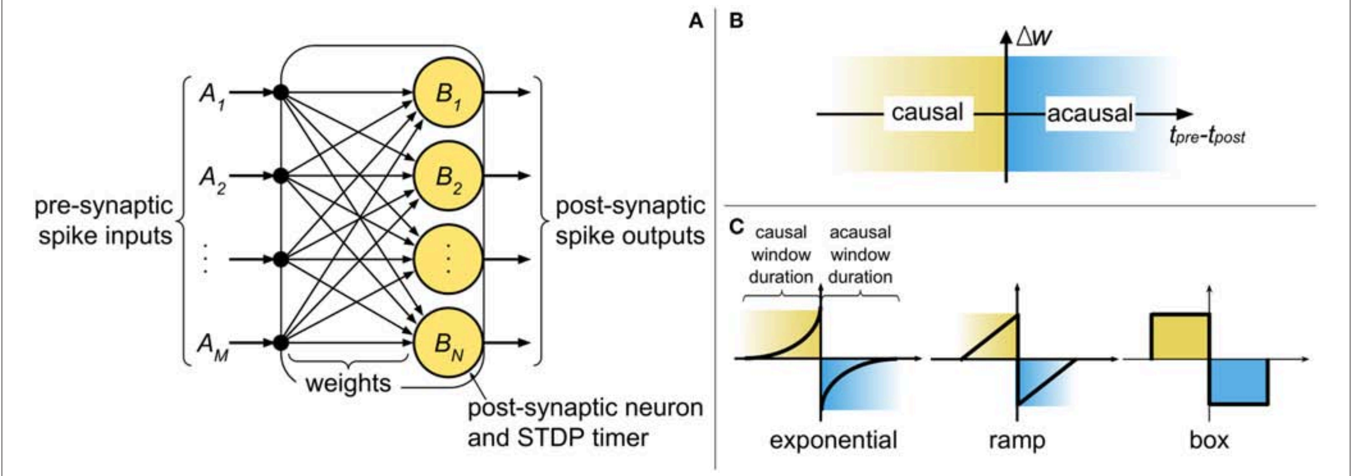


FIGURE 1 | (A) An abstract representation of a neuromorphic spiking neural network core and the components required for implementing pre-synaptic spike-driven STDP. **(B)** The causal and acausal regions of the STDP function. **(C)** Typical STDP kernels implemented in neuromorphic systems.

between the spikes, and can also consider other factors (such as the current weight value). In sum, the polarity of change depends on the order of the spikes, while the intensity of change depends on the temporal difference of the spikes. The basic model for STDP is defined mathematically by

$$\Delta w_{ij} = \sum_{a=1}^{T_j} \sum_{b=1}^{T_i} W(t_j^a - t_i^b), \tag{1}$$

where the weight change between pre-synaptic neuron j and post-synaptic neuron i is defined by the STDP kernel, W, using all T_j pre-synaptic spike times, t_j, and all T_i post-synaptic spike times, t_i.

The STDP kernel is a function which defines how weights are modified based on the relative temporal difference between pre- and post-synaptic spikes. **Figure 1C** highlights the causal (when $t_{pre} < t_{post}$) and acausal (when $t_{pre} > t_{post}$) regions of the STDP function in three commonly used kernels: (truncated) exponential, ramp, and box. The basic STDP model in Equation (1) considers a causal relationship of infinite duration between all pre- and post-synaptic spikes. However, physical realizations of STDP cannot account for a limitless amount of data to be stored and analyzed at every instant of weight update. Therefore, two considerations must be made for temporal spike interaction when implementing STDP in a neuromorphic system: (1) the duration of the kernel is finite and (2) the number of spike times which can be stored is finite. For the first consideration, the typical STDP kernels in **Figure 1C** present finite causal and acausal window duration. In hardware, this duration is defined by the limit of the STDP timers used in the system. The exponential kernel, in theory, has a window duration of infinite time; nonetheless, for physical realizations of the kernel, we define a limit (i.e., truncation) on how far apart in time two spikes can influence weight change. With the ramp and box kernels, this limit is naturally occurring. For simplifying things further, we normally select symmetric kernels (i.e., with identical duration of the causal and acausal windows) as not to require different STDP timers for each side of the STDP kernel. The second consideration affects the temporal spike interaction and is, in part, addressed by the finite kernel duration since "older" spikes (i.e., spikes which have already left the learning window) can be discarded.

Lastly, throughout this paper we will represent the STDP window duration as T_{stdp} and the refractory period duration as T_{refr}. Since we are considering implementations on digital neuromorphic systems, both of these duration values are defined as integer multiples of the system time step, Δt. Additionally, it is worth mentioning that there are basically two alternatives for storing spike times: using a bitmap or using multiple timers. In section A1 we detail how the latter is always at least as efficient as the former and, thus, this will be our method of choice throughout the paper. Nevertheless, the proposed STDP learning method using multiple timers can be transferred seamlessly to a bitmap representation of spike times if desired.

2.3. Synaptic Weight Data Structures

Storage costs associated to synaptic weight memory arrangements have been previously studied (Moradi et al., 2013; Joshi et al., 2017; Kornijcuk et al., 2018), and here we give an overview of four typical data structures used for representing synaptic weights. We analyze the storage costs (in number of bits) based on number of neurons, weight bit-length, and varying degrees of network connectivity density. Depending on the network topology being emulated, particularly with regards to the connectivity density between pre- and post-synaptic neurons, some of the data structures have clear advantages over the more traditional dense matrix representation. The data structures present common memory tables, which include: adjacency table, pointer table, and weight table. Which tables are used and how they are organized defines the synaptic weight memory arrangement of the network.

As will be presented next, crossbars consume memory even for nonexistent synaptic connections, while pointer-based models store only the existent connections, making them ideal candidates when representing sparsely connected networks. For our analyses, the network connectivity density, ρ, represents the percentage of post-synaptic neurons which are connected to a given pre-synaptic neuron, while sparsity can be computed simply as $(1-\rho)$. Both crossbars and pointer-based architectures present a weight table (WT) for storing the values of the synaptic weights; however, the latter must (directly or indirectly) also include in WT the address of the post-synaptic neuron associated with each weight, along with an additional memory called the pointer table (PT).

2.3.1. Fully Connected: Crossbar

The most intuitive representation of synaptic weight memory arrangement is by means of a dense matrix, representing full connectivity between the inputs (pre-synaptic neurons) and outputs (post-synaptic neurons). Alternatively, in neuromorphic systems, the dense matrix is sometimes referred to as a *crossbar* (Merolla et al., 2014). In a crossbar, every connection between a pre- and post-synaptic neuron has a reserved space in WT, even if the connection between the neurons does not exist.

An important aspect of WT to consider is that, when using a dense matrix to represent a sparsely connected network, the zero-valued weights can represent either (1) a nonexistent connection or (2) an existent connection with weight currently equal to zero ("inactive"). When simply testing the network (i.e., while not performing synaptic plasticity), both of these cases produce the same results. However, when actually training the network, there should be a distinction between a nonexistent connection and a weight which can momentarily take on the value of zero. To distinguish between these two cases, the first option is to use an additional memory called the adjacency table (AT), where each position a_{ij} in AT stores a binary value representing the existence ($a_{ij} = 1$) or nonexistence ($a_{ij} = 0$) of the synaptic connection between pre-synaptic neuron A_j and post-synaptic neuron B_i (Joshi et al., 2017). The second option is to use one of the 2^W weight values—where W represents the bit-length of each weight—to represent a nonexistent connection. The advantage of using this second option is that it removes the memory overhead required for storing AT, thus only using one weight value—instead of an additional bit per weight—to differentiate between existent and nonexistent connections. Throughout our work, crossbars will be implemented using this second option.

The top left panel in **Figure 2** depicts a crossbar with M pre-synaptic and N post-synaptic neurons. Though WT can be represented in matrix-form, in the actual memory the weights are stored sequentially, starting with all the weights of pre-synaptic neuron A_1 (i.e., w_{11} to w_{N1}), then all the weights of A_2 (i.e., w_{12} to w_{N2}), and so forth, until weights w_{1M} to w_{NM}. Since the crossbar presents a structured WT, the start and stop locations of the weights in WT for each pre-synaptic neuron can be obtained simply by the pre-synaptic address, thus eliminating the need for pointers: the location of the first weight for pre-synaptic neuron A_j can be computed by $A_j^* = (j-1)N + 1$, with $j \in [1, M]$. Therefore, forward access in crossbars is performed by starting at address $WT(A_j^*)$ and reading N consecutive weights. The figure also illustrates forward access (in yellow) for a single pre-synaptic neuron.

2.3.2. Pointer-Based Compressed Sparse Row (PB-CSR)

Using the compressed sparse row (CSR) format (Saad, 2003), each position of WT stores an address-weight pair, (B_i, w_{ij}), of the post-synaptic neuron B_i and the respective incoming weight from pre-synaptic neuron A_j. In this manner, WT is only populated by existent synaptic connections, and is the most efficient method for storing very sparse networks. The top right panel in **Figure 2** exemplifies the PB-CSR model. As shown in the figure, an important aspect of this model is that, when accessing the weights for pre-synaptic neuron A_j, since we do not have explicit information of the number of existent connections for this neuron, we must always read the start, $PT(j)$, and stop, $PT(j+1)$, addresses. Therefore, for performing forward access of pre-synaptic neuron A_j, start at position $PT(j) = A_j^*$ in WT and consecutively read addresses and weights until position $A_{j+1}^* - 1$. The figure also illustrates the forward path (in yellow) for a single pre-synaptic neuron in PB-CSR, requiring two reads in PT (for start and stop) and ρN reads in WT for the existent connections.

2.3.3. Pointer-Based Run-Length Encoding (PB-RLE)

Run-length encoding (RLE) is a method of lossless data compression particularly useful when consecutive sequences of the same value are present (Oliver, 1952). This concept can be used to replace explicit storage of post-synaptic neuron addresses of adjacent nonexistent connections. In PB-RLE, sequences of consecutive nonexistent connections are stored as run counts, and each position in WT stores a "run bit" followed by the run/weight value. A run bit equal to "0" indicates the existence of the synaptic connection, and the value that follows the bit specifies the respective synaptic weight. If the run bit equals "1," then the data that follows it specifies the run length, representing the number of consecutive post-synaptic neurons which do not have connections with the respective pre-synaptic neuron and are, thus, "skipped" when sequentially reading through WT.

The bottom left panel in **Figure 2** illustrates the PB-RLE model. Since the resulting WT after compression depends on the specific distribution of the existent connections in the network, we included equations for the worst-case scenario of perfectly interleaved runs and weights. In other words, for $\rho < 0.5$, no two consecutive positions in WT contain existent connections; for $\rho \geq 0.5$, no two consecutive connections are nonexistent, resulting in only runs of unit length. The figure also illustrates the forward path (in yellow) for a single pre-synaptic neuron, A_j, which consists on starting at position $PT(j) = A_j^*$ in WT and consecutively reading weights and processing runs until post-synaptic neuron N. When reading the last weight or run, the pointer should be in position $A_{j+1}^* - 1$ in WT. Forward access requires one read in PT and a variable number of reads in WT, which depends on the distribution of connections between the pre- and post-synaptic neurons. The equations in the figure are

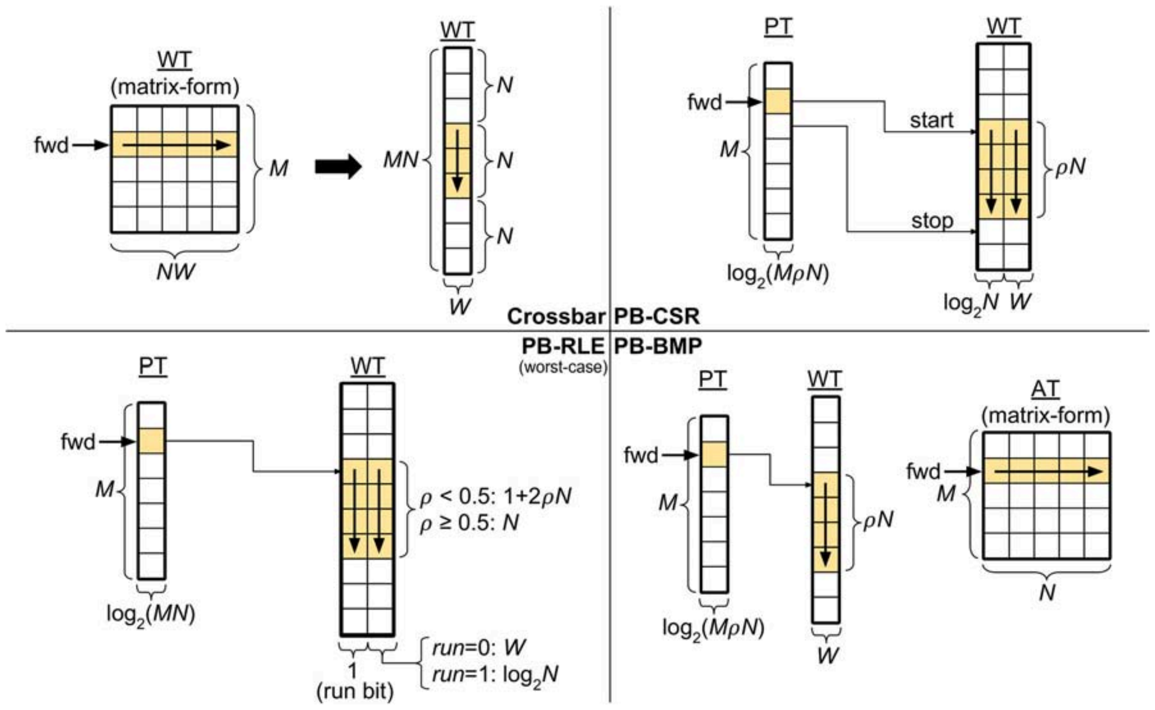


FIGURE 2 | Synaptic weight memory arrangements and storage costs (in bits). Tables: adjacency table (AT), pointer table (PT), and weight table (WT). Parameters: number of pre-synaptic neurons (*M*), number of post-synaptic neurons (*N*), weight bits (*W*), and connectivity density (ρ). The forward memory access path has been highlighted. The pointer-based data structures compress data storage, resulting in non-structured solutions which depend on the connectivity and weight distribution in the network. The equations for PB-RLE refer to worst-case scenarios of perfectly interleaved runs and weights.

defined for the worst-case scenario of perfectly interleaved runs and weights.

2.3.4. Pointer-Based Bitmap (PB-BMP)

Mixing properties of the crossbar and the previous pointer-based data structures, the PB-BMP includes PT, WT, and an additional fully connected adjacency table. As with PB-RLE, bitmaps do not require explicit storage of post-synaptic neuron addresses in WT, while its equivalent run-length encoding is realized via AT. The bottom right panel in **Figure 2** illustrates the PB-BMP model and the forward access path (in yellow) for a single pre-synaptic neuron. The start address is stored in PT, and AT stores binary information about connection existence. For forward access of pre-synaptic neuron A_j, start the pointer in WT at position $PT(j) = A_j^*$, and in matrix-form AT continuously read the entire row j in the following manner: for every position in AT which $a_{ij} = 1$, read the current weight in WT and move the pointer in WT to the next position; if $a_{ij} = 0$, do not change the pointer in WT. After reading the entire row j in AT, the pointer in WT should be at position A_{j+1}^*. The entire forward access requires one read in PT, N reads in AT, and ρN reads in WT.

2.3.5. Data Structure Storage Costs

When considering a complete neuromorphic system, memory elements must also be accounted for storing neuron variables (e.g., synaptic current, membrane potential, etc.) and the aforementioned STDP timers. However, for a network with k pre-synaptic and k post-synaptic neurons, the space complexity of storing the synaptic weights is $\mathcal{O}(k^2)$, while neuron variables and timers are unique to each neuron and do not depend on the synaptic weight memory arrangement being used, resulting in $\mathcal{O}(k)$ space complexity. Therefore, our analyses of memory storage cost and efficiency only incorporate the memory required for storing pointer, adjacency and weight tables, and do not account for the neuron variables and STDP timers.

A summary of the storage costs (in number of bits) for the different synaptic weight memory arrangements is presented in **Table 1**. The crossbar does not require AT since one of the 2^W weight values can be used to indicate nonexistent connections. The upper limit of PB-RLE costs vary depending on connectivity density: for $\rho < 0.5$ we considered no two consecutive existent connections, while for $\rho \geq 0.5$ we considered every run is of unit length. Actual costs for PB-RLE (presented in **Figure 6**) were obtained via simulation, where networks were generated by randomly creating connections based on the value of ρ, then producing the respective PT and WT and computing their costs in terms of number of bits required for storage.

2.3.6. Data Structure Access Costs

Both forward and reverse access to synaptic connections are required for implementing the original STDP learning rule. When a pre-synaptic neuron spikes, we perform forward access in the connectivity table and apply the acausal updates, since this specific pre-synaptic spike must have occurred *after* any post-synaptic spikes which have already taken place. When a post-synaptic neuron spikes, we perform reverse access in the connectivity table and apply the causal updates, since any pre-synaptic spike must have occurred *before* this specific post-synaptic spike.

In the diagrams in **Figure 2**, the forward ("fwd") path for accessing weights from pre- to post-synaptic neurons in the weight tables was highlighted in yellow. The structured memory arrangement in crossbars facilitates reverse access by simply performing forward access in the transposed WT. Due to the manner in which weights are stored in memory, pointer-based data structures natively present access only to forward connectivity. For accessing post-to-pre connections (i.e., reverse access), two alternatives are possible: (1) using forward access and sweeping through the entire AT or WT to verify if each pre-synaptic neuron is connected to the post-synaptic neuron of interest or (2) including PT and WT for the reverse connections as well. The first solution does not affect hardware costs, but can be extremely inefficient in terms of computation time (particularly for densely connected networks). The second solution facilitates reverse access by creating explicit tables for this purpose, yet at the cost of basically doubling the memory requirements. In this subsection we will only treat the first option since the second option can be trivially implemented by simply executing forward access on the reverse tables. A final alternative will be presented in section 2.4, where we describe how STDP learning can actually be executed without the need for reverse access, availing of the benefits of pointer-based models (i.e., memory compression and efficient forward access).

An important practical aspect to consider is that memory access in digital memory elements, such as double data rate synchronous dynamic random-access memory (DDR SDRAM), typically occurs in blocks of multiple bytes per read command. Additionally, there is a variable amount of row and column address strobe overhead that precedes the single memory access depending on whether the read is from the same row or from the next column item. For single item accesses, this can add many clock cycles of overhead for reading. Memory controllers can try to optimize memory command scheduling to overcome some of this, but never all of it. Nonetheless, for simplification purposes, in our work we have considered that accessing any single position in memory (to read the value of a single variable) consumes one "computational unit," and that only one position in memory can be accessed at a time. With this, the computational (or access) cost of performing STDP can be summarized simply by the number of positions in memory which must be accessed to obtain address and weight information for executing the learning rule.

A summary of the access costs for the different synaptic weight data structures is presented in **Table 2**. In the table, forward costs refer to the average number of positions in the data that must be accessed for a single pre-synaptic neuron, while reverse costs refers to the average number of positions in the data that must be accessed for a single post-synaptic neuron. The equations in the table consider worst-case scenarios for PB-RLE in forward access, as well as worst-case scenarios for all pointer-based data structures in reverse access. Exact closed-form solutions, particularly for reverse access, are difficult to obtain for pointer-based models since the location and distribution of existent connections can greatly impact the data compression, consequently affecting the search for addresses and weights. In any case, since our proposed method removes reverse access altogether, we will focus uniquely on forward access throughout the paper, with the equations in the table merely serving as an assessment of the complexity of reverse access.

2.4. STDP Learning Rule With Forward-Only Connectivity Access

Based on the equations presented in **Table 2**, reverse access in pointer-based data structures can be quite inefficient. Because of this limitation, multiple efforts have been made in approximating STDP learning using forward-only connectivity, including simplifying the STDP rule by equally updating all the synaptic weights based on recent spike activity, using other variables as a proxy for the post-synaptic spike times when computing causal updates, and delaying the weight updates. Our method falls under the latter category; however, contrary to these approximate alternatives, it can produce exact equivalence to STDP, as will be shown in the Results section.

When using pointer-based data structures for storing synaptic weights, acausal updates can be immediately performed at

TABLE 1 | Storage costs (in bits) for different synaptic weight memory arrangements.

Architecture		AT	PT	WT
Crossbar[a]		0	0	MNW
PB-CSR		0	$M\log_2(M\rho N)$	$M\rho N(\log_2 N + W)$
PB-RLE[b]	$\rho < 0.5$	0	$M\log_2(MN)$	$M\rho N(2+\log_2 N + W) + M\log_2 N$
	$\rho \geq 0.5$		$M\log_2(MN)$	$MN(1 + (1-\rho)\log_2 N + \rho W)$
PB-BMP		MN	$M\log_2(M\rho N)$	$M\rho NW$

[a]The crossbar does not require AT since one of the 2^W weight values will be used to indicate nonexistent connections.

[b]This is the upper limit of the cost, considering perfectly interleaved runs and weights. More realistic values were obtained via simulation.

TABLE 2 | Access costs (per neuron) for different synaptic weight memory arrangements.

Direction	Architecture		AT	PT	WT
Forward	Crossbar		0	0	N
	PB-CSR		0	2	ρN
	PB-RLE[b]	$\rho < 0.5$	0		$1 + 2\rho N$
		$\rho \geq 0.5$		1	N
	PB-BMP		N	1	ρN
Reverse[a]	Crossbar		0	0	M
	PB-CSR		0	M	$M(\rho N)$
	PB-RLE	$\rho < 0.5$	0		$M(1 + 2\rho N)$
		$\rho \geq 0.5$		M	MN
	PB-BMP		$M + \rho M(N-1)$	ρM	ρM

[a]The equations for the pointer-based models consider worst-case scenarios. The values presented in ***Figure 6*** were obtained via simulation.

[b]This is the upper limit of the cost, considering perfectly interleaved runs and weights. More realistic values were obtained via simulation.

the onset of a pre-synaptic spike using forward connectivity access of PT. Causal STDP updates, however, should be performed at the onset of post-synaptic spikes, requiring reverse connectivity access. Since pointer-based models natively have only forward connectivity access, we have devised a method which performs causal updates at the onset of yet another pre-synaptic neuron event: the STDP timer expiration. Therefore, instead of immediately applying the causal updates at the onset of post-synaptic spikes, the update is delayed until the pre-synaptic STDP timer expires, at which point the causal influence of a spike ceases. The two types of weight updates in our proposed algorithm are described below:

- **Acausal update:** At the onset of a pre-synaptic spike from neuron A_j, perform forward access in WT starting at position $PT(j) = A_j^*$, and verify the STDP timers of the post-synaptic neurons connected to A_j. For every post-synaptic neuron which has spiked not long ago (i.e., with an active STPD timer), perform the acausal weight update.
- **Causal update:** At the moment of expiration of the pre-synaptic STDP timer of neuron A_j, perform another forward access in WT starting at position $PT(j) = A_j^*$, once again verifying the STDP timers of the post-synaptic neurons connected to A_j. For every post-synaptic neuron which has recently spiked (i.e., with an active STPD timer), perform the causal weight update.

For clarifying the proposed algorithm, **Figure 3** illustrates four different instants during system evolution for a causal and an acausal STDP window duration of 8 time steps each. These events are described below:

1. The first event illustrates a new post-synaptic spike, at which time this neuron's STDP timer is initialized and no weight updates occur.
2. In the second event, the pre-synaptic neuron elicits a new spike, initializing its STDP timer and also performing the acausal weight update. This update is performed just as it would be in the original STDP algorithm via forward connectivity access.
3. The third event illustrates the expiration of the post-synaptic STDP timer. No action is required since the acausal update of its weight has already been serviced.
4. In the fourth event, the pre-synaptic STDP timer expires, at which point the causal weight update takes place. Unlike the original STDP algorithm, in which causal updates would have taken place at the onset of a post-synaptic spike, the proposed method delays the update until the pre-synaptic STDP timer expires, requiring, therefore, only a second forward access and avoiding reverse connectivity access altogether.

Using our method, if every neuron is configured to be able to spike at most once during the STDP window, then the weight updates will always fall under one of these four scenarios and produce results which exactly match those obtained by the original STDP algorithm (this will be shown in section 3.3). However, if a neuron is allowed to spike multiple times during T_{stdp}, then many different scenarios may arise between the moment a post-synaptic neuron spikes and the moment the STDP timer of its pre-synaptic neuron expires. In this case, the proposed method may incur in incorrect weight updates, as shown next.

2.4.1. Drawbacks of Allowing Multiple Spikes Inside the STDP Window

If the system is designed without guaranteeing that no neuron spikes more than once inside its STDP window, some natural drawbacks arise. Below we list these cases to better illustrate the importance of the two criteria— three of the drawbacks present direct solutions, while the fourth does not. To generate these specific cases, we will consider nearest-neighbor temporal spike interaction (where only the nearest spikes are considered; refer to subsection 2.4.3), and we will configure the neurons with $T_{refr} < T_{stdp}$ and use a single timer of length $\lceil \log_2(T_{stdp} + 1) \rceil$ bits per neuron.

Case 1: High-firing pre-synaptic neuron (refer to Figure 4A): If a second pre-synaptic spike occurs while the first spike is still inside the STDP window, the timer will be restarted and information about the first spike will be lost. Since the

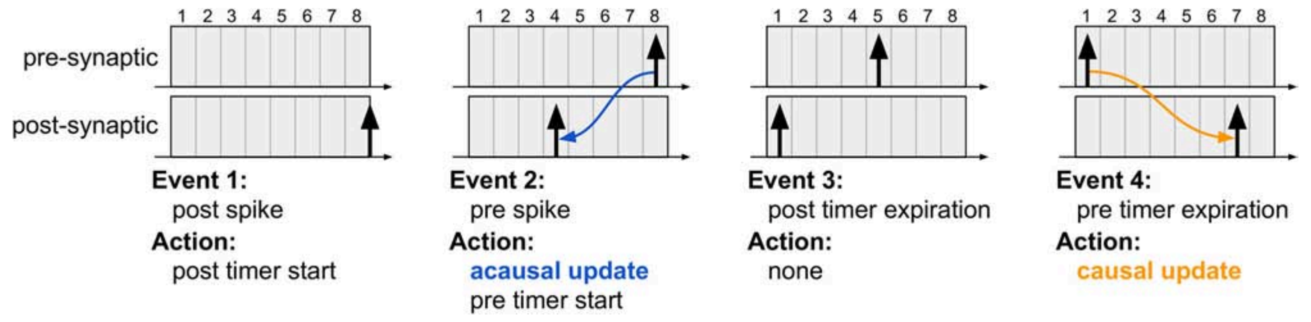


FIGURE 3 | The four typical events which occur during the proposed STDP learning algorithm. The first event illustrates post-synaptic spike generation, while the third event is the moment a post-synaptic spike exists the learning window (i.e., its STDP timer expires); in both cases, no weight updates are performed since the algorithm is driven only by pre-synaptic events. The second event illustrates pre-synaptic spike generation, resulting in acausal update. The fourth event illustrates pre-synaptic STDP timer expiration, resulting in causal update. Note that the post-synaptic spikes in the third and fourth instants are distinct spike events, used to highlight that acausal and causal updates can take place between the same pair of neurons depending on the order of the spikes.

post-synaptic spikes occur after the second pre-synaptic spike, the correct update will take place since only nearest-neighbor influence is considered.

Case 2: High-firing post-synaptic neuron (refer to Figure 4B): If a second post-synaptic spike occurs before the pre-synaptic spike, information about its first spike time will be lost. Since the pre-synaptic spike occurs after the second post-synaptic spike, once again the correct update will take place since only nearest-neighbor influence is considered.

Case 3: High-firing pre-synaptic neuron (refer to Figure 4C): If a second spike occurs for a pre-synaptic neuron whose STDP timer has not yet expired, then the timer will be restarted and information about the first spike will be lost. As a solution, first service the pending causal updates (relative to the first spike), then service the acausal updates (relative to the second spike) only for post-synaptic spikes which have occurred after the first pre-synaptic spike. The reason for this is that the acausal updates of post-synaptic spikes older than the first pre-synaptic spike have already been performed at the onset of this first spike. Lastly, restart the STDP timer for the new spike.

Case 4: High-firing post-synaptic neuron (refer to Figure 4D): If we have a post-synaptic neuron which spikes frequently (i.e., before the pre-synaptic timer expires and the causal updates are performed), then the nearest-neighbor spike information between pre- and post-synaptic neurons will be lost and overwritten by the new post-synaptic spike time (since the post-synaptic STDP timer is restarted). An objective, yet inexact, solution is to simply ignore this issue given that a single pre-synaptic spike should not have a strong causal relation with a high-firing post-synaptic neuron. With this, a causal update will still take place at the expiration of the pre-synaptic STDP timer, except it will just not be with the nearest-neighbor post-synaptic spike. To prevent this scenario from occurring, we must ensure that a maximum of a single spike can occur in the duration of each timer, demanding that the system be designed as presented next.

2.4.2. Criteria for Exactness Between Methods

The effect of not being able to implement nearest-neighbor causal updates has the effect of the weights not increasing as much as expected, resulting in lower synaptic efficacy and, consequently, fewer post-synaptic spikes. For the results of the proposed method to exactly match those obtained by the original STDP algorithm, each neuron must present one timer per refractory period, capturing every possible spike, and resulting possibly in multiple timers to cover the entire duration of the STDP learning window. In other words, we must use $\lceil T_{stdp} / T_{refr} \rceil$ timers, each of length $\lceil \log_2(T_{refr} + 1) \rceil$ bits. Note that if $T_{refr} \geq T_{stdp}$, this reduces to the expected single timer of length $\lceil \log_2(T_{refr} + 1) \rceil$ bits. This rule has the advantage of allowing different types of temporal spike interaction (see subsection 2.4.3).

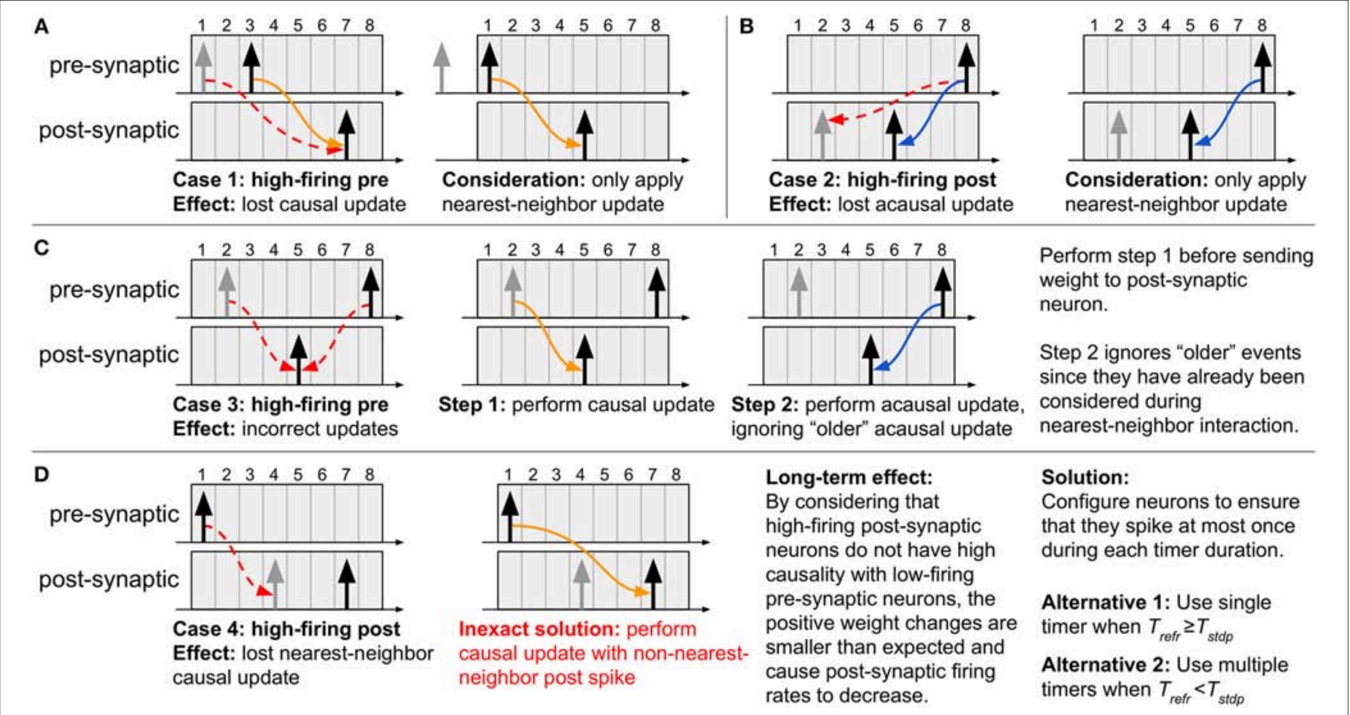

A
1 2 3 4 5 6 7 8
pre-synaptic
post-synaptic
Case 1: high-firing pre
Effect: lost causal update
Consideration: only apply nearest-neighbor update
B
Case 2: high-firing post
Effect: lost acausal update
Consideration: only apply nearest-neighbor update
C
Case 3: high-firing pre
Effect: incorrect updates
Step 1: perform causal update
Step 2: perform acausal update, ignoring "older" acausal update
Perform step 1 before sending weight to post-synaptic neuron.
Step 2 ignores "older" events since they have already been considered during nearest-neighbor interaction.
D
Case 4: high-firing post
Effect: lost nearest-neighbor causal update
Inexact solution: perform causal update with non-nearest-neighbor post spike
Long-term effect: By considering that high-firing post-synaptic neurons do not have high causality with low-firing pre-synaptic neurons, the positive weight changes are smaller than expected and cause post-synaptic firing rates to decrease.
Solution: Configure neurons to ensure that they spike at most once during each timer duration.
Alternative 1: Use single timer when $T_{refr} \geq T_{stdp}$
Alternative 2: Use multiple timers when $T_{refr} < T_{stdp}$

FIGURE 4 | Special cases which arise when using a single timer and $T_{refr} < T_{stdp}$. **(A–C)** By considering nearest-neighbor temporal spike interaction, cases 1–3 can be correctly addressed, **(D)** yet case 4 does not present a direct solution. To overcome all the drawbacks inherent to the proposed method, the neurons in the system must be configured as to ensure that they spike at most once during each timer duration. If the neurons can be configured with $T_{refr} \geq T_{stdp}$, then we guarantee that only one spike can occur inside the STDP window. However, if the neurons present $T_{refr} < T_{stdp}$, then the only manner of capturing all the spikes is to use multiple timers for the STDP window, each with duration T_{refr}.

Details of the multi-timer method are presented in **Appendix A1** and shown in Figure A1. To implement our proposed method of STDP learning using multiple timers, we must simply treat each individual timer as was done in **Figure 3**. The causal updates, however, can be implemented in two different manners.

1. During the traversal of the spike through the timers, at the instant of timer expiration the causal updates are performed between the current spike and all "newer" post-synaptic spikes. This means that whenever any of the multiple pre-synaptic timers expires, perform weight updates with the post-synaptic spikes which have recently entered the queue—meaning we must verify only the first timer of the post-synaptic neurons.
2. The second option implies in performing the causal update only when the T-th (i.e., the last) pre-synaptic timer expires. This method has the advantage of possibly incurring only two instants of updates: when the spike enters and when it exists the spike history queue. However, if a new pre-synaptic spike occurs while a spike is still traversing the queue, then the causal weight updates between the first spike and any post-synaptic spikes that occurred after it must be performed prior to updating the post-synaptic neuron variables. This effect is similar to that of case 3 in **Figure 4**.

It may appear at first glance that both of these alternatives incur in more memory access than the original STDP algorithm. The first method can, in fact, produce more updates than the second alternative, particularly for sparse pre-synaptic activity—though it is a more systematic way of implementing updates since we must only verify the first timers for the post-synaptic neurons. The second alternative, however, implements updates only when actually required, consuming (on average) the same number of memory accesses as the original STDP learning rule. This can be elucidated by considering the case of a high-firing post-synaptic neuron: the original algorithm would search through all its pre-synaptic neurons even if most have not spiked, while the proposed algorithm would only verify the pre-synaptic neurons which have recently spiked and could, therefore, have some causal influence on the post-synaptic spikes. If we consider the case of a high-firing pre-synaptic neuron, then the inverse is valid, thus resulting most likely in a similar average cost for both methods.

2.4.3. Temporal Spike Interaction

Temporal spike interaction can go to the extreme of considering only the nearest spikes, known as *nearest-neighbor interaction* (Morrison et al., 2008). At the other extreme, *all-to-all interaction* considers influence of the entire spike history. A third variant is a *triplet-based interaction* (Pfister and Gerstner, 2006), where a sequence of post-pre-post spikes, for example, is a template for updating weights. Examples illustrating these temporal spike interactions using multiple timers for $T_{stdp} = 12$ and $T_{refr} = 5$ are presented in **Figure 5**. The procedure when using multiple timers follows that of a single timer: weights are updated at the onset of a new pre-synaptic spike and at the expiration of the (last) pre-synaptic STDP timer. Note in **Figure 5C** that the triplet-based interaction requires spikes to be stored for a longer duration since the "older" post-synaptic spike in the post-pre-post triplet may already have left its active region (i.e., the timers to the right of the red bar), but is still of use for an active pre-synaptic spike. From the figure we show that, independently of the type of temporal spike interaction being implemented, as long as the appropriate number of timers is used and we address the pending causal updates before sending the weights to the post-synaptic neurons (as per case 3 in **Figure 4C**), then our method produces exact equivalent results to original STDP.

3. RESULTS

3.1. Data Structure Efficiency

Based on the data structure storage and access costs, a comparison of storage and forward access efficiencies for multiple network sizes, weight bit-lengths, and connectivity densities is shown in **Figure 6**. By varying the number of pre-synaptic (M) and post-synaptic (N) neurons, the connectivity density (ρ), and the number of bits used to represent each weight (W), we empirically verified the performance of each data structure for different network configurations. For each data structure, storage cost, C_s, is compared to the reference cost value, $C_s^{ref} = M\rho NW$, representing the amount of memory required to store the weights of only the existent connections in the network. Storage efficiency is then computed as $\eta_s = C_s^{ref}/C_s$. Forward access cost, C_a, is compared to the reference computational cost value, $C_a^{ref} = \rho MN$, representing the total number of variables to be accessed when reading data for all pre-synaptic neurons once (i.e., obtaining the entire network address-weight pairs). Forward access efficiency is then computed as $\eta_a = C_a^{ref}/C_a$. The results in the plots were obtained by generating 1,000 randomly connected networks according to the parameter set, and averaging the costs of these networks per connectivity density. The light-shaded regions behind each plot indicate the model with the highest efficiency for specific values of ρ.

As we can observe in **Figure 6A**, pointer-based models have a great advantage over crossbars due to their data compression, with the PB-BMP model showing the best overall performance for a large range of ρ. Naturally, for larger weights, pointer-based models show a greater advantage, particularly for sparsely connected networks (i.e., small values of ρ). Increasing network size has only a slight impact on PB-BMP models, since in these models the only additional memory required beyond the reference value is the rather low-cost AT. Conversely, PB-CSR and PB-RLE are clearly affected when mapping larger networks since they directly (for PB-CSR) or indirectly (in run-lengths for PB-RLE) must store larger post-synaptic addresses in WT. For forward access, **Figure 6B** shows that pointer-based models PB-CSR and PB-RLE have a natural advantage over the other two models since they do not require reading every position in their tables. Between these two models, PB-CSR performs better than PB-RLE (except for $\rho = 1$) because the latter requires decompressing the data by reading run-lengths, while the former requires only two read commands in PT (the start and stop addresses) along with the ρMN weights to be read. The PB-BMP

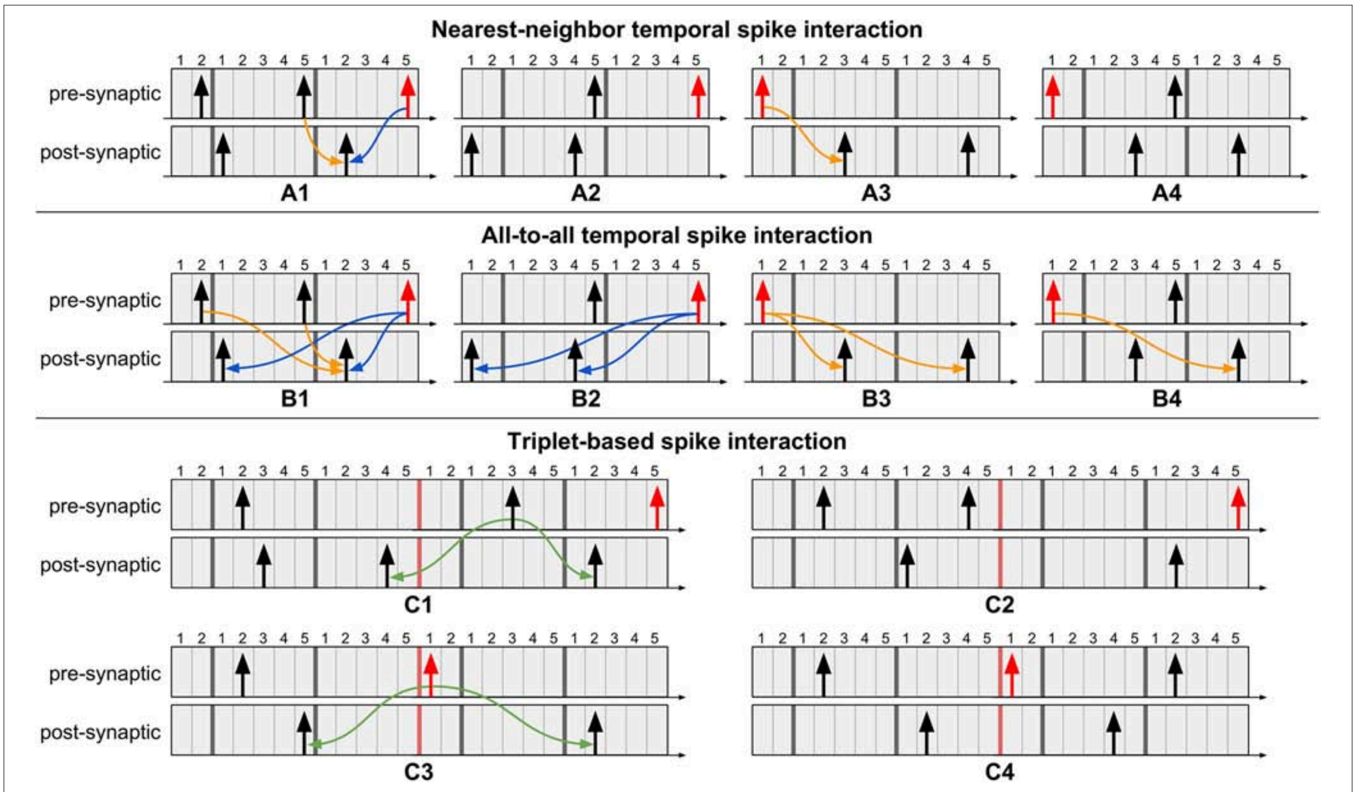

Nearest-neighbor temporal spike interaction
pre-synaptic
post-synaptic
A1
A2
A3
A4
All-to-all temporal spike interaction
B1
B2
B3
B4
Triplet-based spike interaction
C1
C2
C3
C4
1 2 1 2 3 4 5 1 2 3 4 5

FIGURE 5 | Example scenarios of proposed method for different types of temporal spike interactions using multiple timers. Pre-synaptic spikes in red represent event of interest: a new spike event or the last STDP timer expiration event. **(A1)** Perform pending causal update before acausal update. **(A2)** No updates required since no post-synaptic spikes occurred after the previous pre-synaptic spike. **(A3)** Perform causal update. **(A4)** No update required because the causal update was performed when the most recent pre-synaptic spike occurred. **(B1)** Perform causal updates with post-synaptic spikes which occurred after the previous pre-synaptic spike, followed by the acausal updates. **(B2)** No causal updates required since no post-synaptic spikes occurred after the previous pre-synaptic spike. **(B3)** Perform causal updates. **(B4)** Only a single causal update is required since the other causal update was performed when the most recent pre-synaptic spike occurred. **(C)** Triplet-based spike interaction requires spikes to be stored for a longer duration. For pre-synaptic spikes, the timers to the right of the red bar represent the active region. **(C1)** Perform triplet update since the previous pre-synaptic spike is still in the active region. **(C2)** No triplet update required since the previous pre-synaptic spike is already in the inactive region (i.e., left-side timers). **(C3)** Perform triplet update at expiration of pre-synaptic STDP timer. **(C4)** No triplet update required since it was performed when the most recent pre-synaptic spike occurred.

model can achieve a maximum efficiency of about 50% because it requires two read commands per existent connection: one read in AT to identify if the connection exists and one read in WT to find the weight value of the connection. The performance of the crossbar grows linearly with connectivity density, and is efficient at very large values of ρ.

3.2. Budget Efficiency

In order to identify the optimal solution for a given implementation budget in terms of memory storage and computational effort (i.e., memory accesses), we defined the *budget efficiency* metric as $\eta = \lambda\eta_s + (1 - \lambda)\eta_a$, where η_s is storage efficiency, η_a is forward access efficiency, and λ is a tunable parameter defining the storage-versus-access trade-off. Note that η_a is computed as the forward access efficiency since (1) both causal and acausal updates only require this type of access in pointer-based models and (2) reverse access in crossbars is just as efficient as forward access.

The graphs in **Figure 7** illustrate the optimal models (based on the shaded colors) for different network parameter settings in the $\rho\lambda$-plane. For networks where memory access efficiency is priority (i.e., small values of λ) and/or for sparse networks (i.e., small values of ρ), the PB-CSR model is the clear optimal solution. This is mainly due to the compression method in PB-CSR, where no AT and no decompression (as in PB-RLE) are required, making weight storage simple and forward access efficient. However, when memory storage is priority (i.e., for large values of λ), the PB-BMP model spans the longest range of connectivity densities as the optimal solution. For densely connected models, the crossbar appears as the best alternative since the nonexistent connections entail only a small amount of storage overhead, while presenting efficient forward access. Interestingly, the PB-RLE model spans only a small region close to the center of the graph (especially for small weight bit-lengths), resulting as the optimal solution for more specific cases of ρ and λ.

3.3. Proof-of-Concept Example

Many of the examples and results presented thus far throughout our work were obtained via simulation of various network

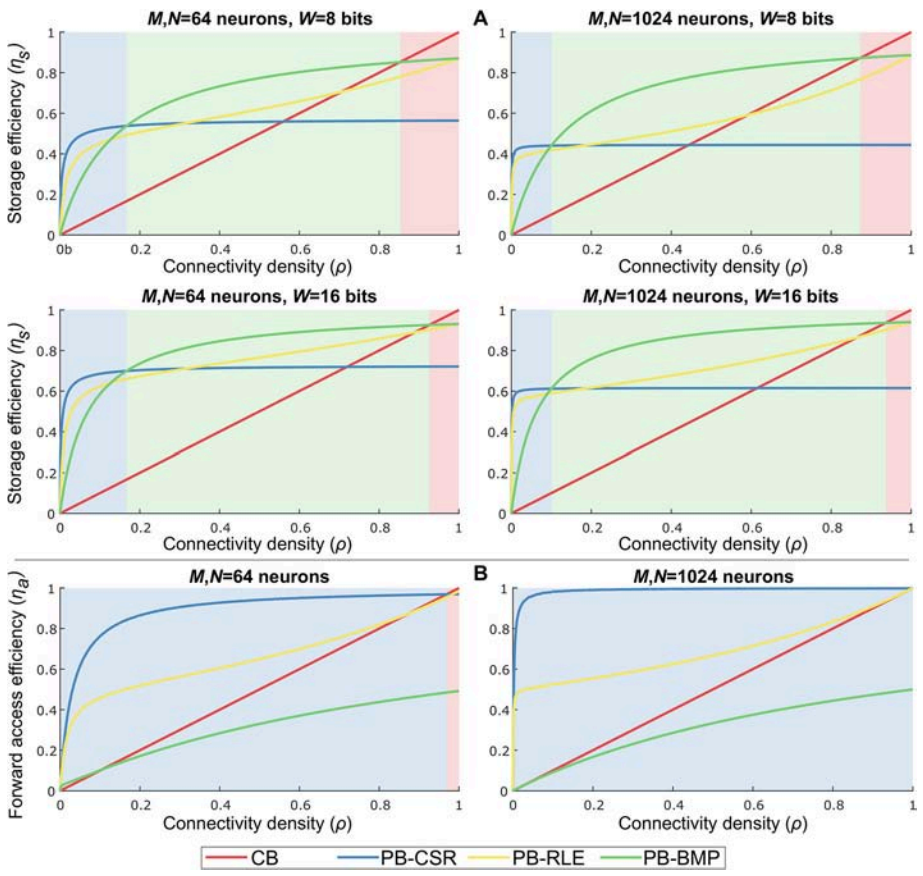


FIGURE 6 | Data structure storage and forward access efficiencies for different parameter settings and varying connectivity density. Parameters: number of pre-synaptic neurons (*M*), number of post-synaptic neurons (*N*), bits per weight (*W*), and connectivity density (ρ). The light-shaded regions behind each plot indicate the most efficient model for specific values of ρ. **(A)** Storage efficiency in pointer-based models is higher than in crossbars for nearly all values of ρ. Increasing weight bit-length is more impactful than increasing network size. PB-BMP efficiency improves for larger networks since it does not explicitly store post-synaptic addresses (except as binary values in AT). All data structures show higher efficiency for larger networks. **(B)** Forward access is efficiently performed in pointer-based models due to their compression mechanism, particularly in sparsely connected networks. PB-CSR has advantage over the other models throughout most values of ρ because it does not require data decompression to obtain address-weight pairs.

topologies and connectivity distributions. In this section, we present an additional example to highlight the equivalence of our proposed algorithm with the original STDP learning rule—when implementing one of the two criteria presented in subsection 2.4.2. The effect of case 4 from subsection 2.4.1— where nearest-neighbor causal updates are lost—will be demonstrated, along with an example of all-to-all temporal spike interaction which perfectly matches the original STDP algorithm.

The experimental setup involves 256 post-synaptic neurons receiving spike inputs from 256 pre-synaptic neurons. Initial weight values were sampled from a Gaussian distribution with 0.1 mean and unit variance. All the neurons were configured with symmetric STDP ramp kernel of window duration of $T_{stdp} = 16$ and maximum weight change of ± 0.01, spiking threshold of $V_{th} = 1.0$, and refractory period duration of $T_{refr} = 4$. Pre-synaptic neurons were set with spiking probability of 10% when outside the refractory period. The leaky integrate-and-fire neuron model was used for the post-synaptic neurons, governed by the equation $V_i(t+1) = \alpha V_i(t) + \sum_j w_{ij} s_j(t)$, where the membrane memory constant, α was set to 0.9. The network dynamics were simulated for 1,000 time steps, during which all the weights and membrane potentials were recorded at each time step. Since causal weight updates occur at different instants of the algorithm for the original STDP learning rule and our proposed method, directly observing the weight values at each time step for such a large number of weights is not feasible. Therefore, to validate our method, we compared the post-synaptic membrane potentials for each neuron throughout the entire simulation. Additionally, for completeness, the post-synaptic spiking activity was analyzed by computing the distance between the van Rossum spike traces (Rossum, 2001) for the two algorithms. The time constant of the exponential kernel for generating the continuous traces was set as the time constant of the membrane potential and computed as $\tau_R = -1 / \log(\alpha) \approx 9.5$.

The simulation results for the network are presented in **Figure 8**, where we verify the convergence of our proposed method for STDP learning. The left column illustrates results when one timer is used and simply nearest-neighbor interaction

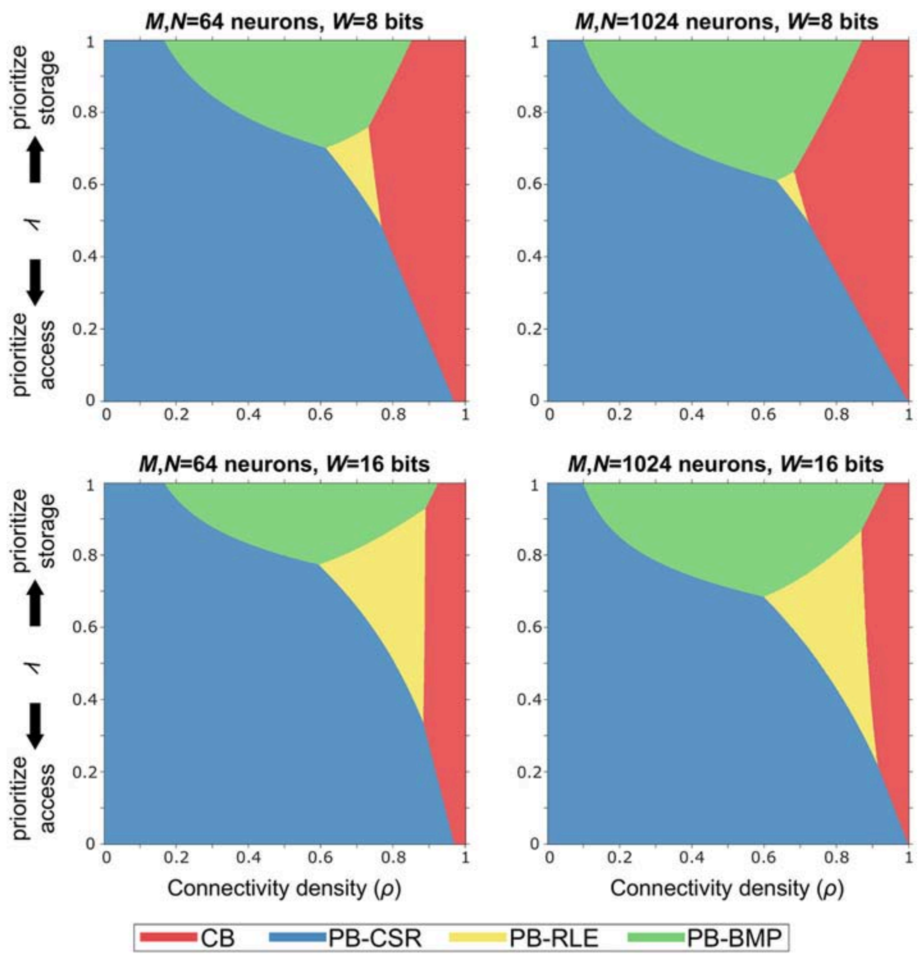


FIGURE 7 | Budget efficiency, $\eta = \lambda\eta_S + (1 - \lambda)\eta_a$. Parameters: η_S is the storage efficiency, η_a is the forward access efficiency, and λ is a tunable parameter defining the storage-versus-access trade-off. Given efficiency priority and overall network connectivity density, the optimal memory arrangement for synaptic weights can be obtained. Pointer-based models cover most of the range of values for ρ and λ because the proposed STDP algorithm takes advantages of their efficient memory compression and forward access.

is considered for the original algorithm and our method. The right column illustrates results when multiple timers (in this case, 4 timers) are used to capture all possible spikes which can occur inside the STDP window and all-to-all spike interaction is performed for the original algorithm and our method. Note that the single-timer and multi-timer results were obtained from different simulations since only one temporal spike interaction can be considered at a time.

The top row shows how the total mean squared error (MSE) of all post-synaptic membrane potentials between the original STDP algorithm and our method diverge when using only one timer; this is the effect described in case 4 in subsection 2.4.1, where post-synaptic weights receive smaller causal updates than expected. For the multi-timer solution, the membrane potentials always match those obtained by the original STDP algorithm, and the resulting MSE is zero.

The second row shows the total van Rossum spike traces obtained by adding all traces after passing each spike through the exponential kernel. In this example, the effect of smaller weight updates because of lost causal nearest-neighbor updates is clearly observed by the decreasing post-synaptic spike activity when using a single timer. As expected, the multi-timer solution produces post-synaptic spikes identical to those obtained by the original STDP algorithm.

Lastly, the bottom row illustrates the MSE of all incoming weights for post-synaptic neuron B_1. Once again, the effect of case 4 causes the weights to diverge for the single-timer solution. For the multi-timer solution, we can see that the MSE momentarily increases but soon after returns to zero; this effect occurs because of the delayed causal updates, but always produces the correct weight at the moment the weight must be effectively used. Note in the graphs that in the last T_{stdp} time steps the membrane potentials and spike traces for the single timer method also converge to zero, simply because we enforced all pre-synaptic neurons to stop spiking during this duration for the final weights obtained by the multi-timer solution to exactly match those of

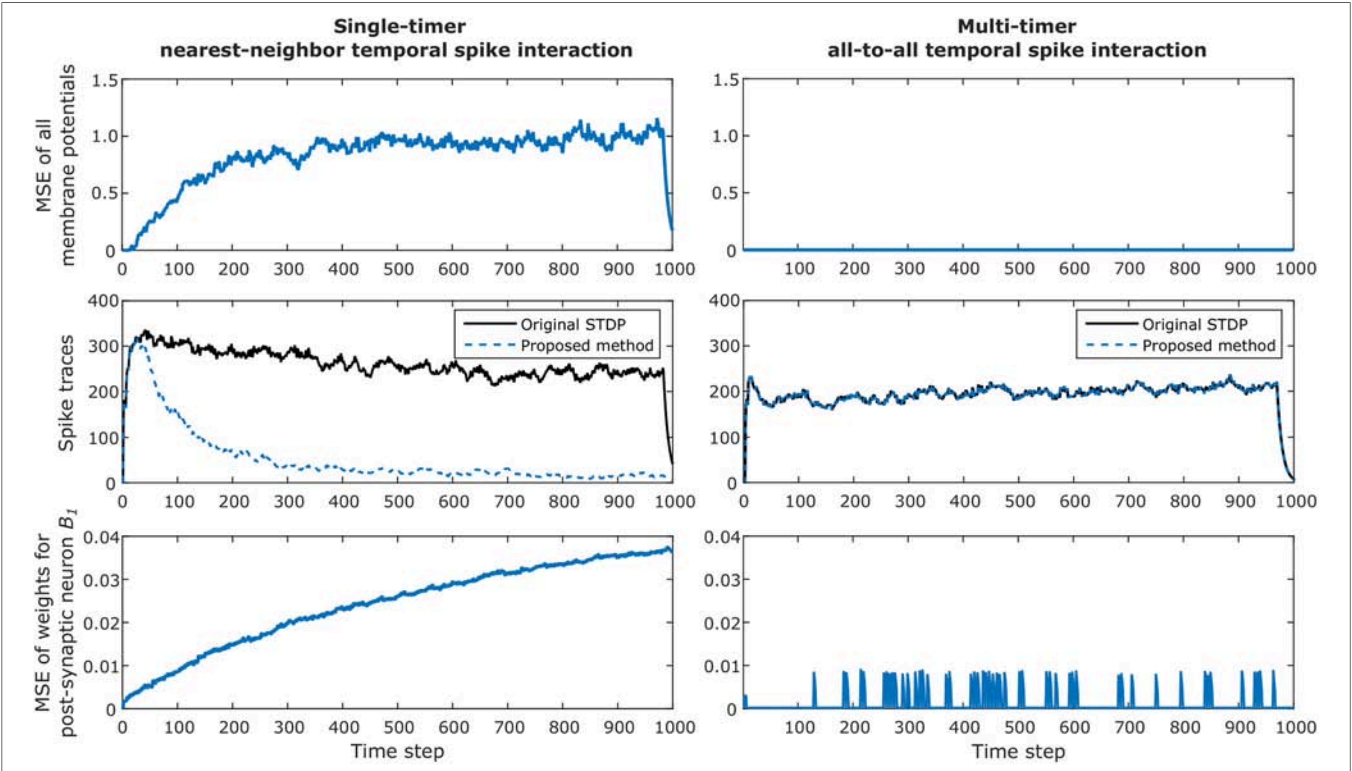


FIGURE 8 | Example of convergence of proposed method for STDP learning for a network with 256 pre-synaptic and 256 post-synaptic neurons, configured with $T_{stdp} = 16$ and $T_{refr} = 4$. The left column illustrates results when one timer is used and simply nearest-neighbor interaction is considered. The right column illustrates results when multiple timers (in this case, 4 timers) are used to capture all possible spikes which can occur inside the STDP window. The top row shows how the total mean squared error (MSE) of all post-synaptic membrane potentials between the original STDP algorithm and our method diverges when using only one timer; this is caused by the effect described in case 4 in section 2.4.1. For the multi-timer solution, the membrane potentials always match, and the resulting MSE is zero. The second row shows the continuous van Rossum spike traces, where the effect of smaller weight updates in the case of using a single timer is clearly observed by the decreasing post-synaptic spike activity over time. As expected, the multi-timer solution produces post-synaptic spikes identical to those obtained by the original STDP algorithm. The bottom row illustrates the MSE of all incoming weights for post-synaptic neuron B_1. Once again the lost causal nearest-neighbor updates make the weights diverge for the single timer solution; for the multi-timer solution, we can see that the MSE temporarily increases but soon after returns to zero, which is simply the effect of the delayed causal updates.

the original STDP algorithm at the last simulation time step (i.e., so the delayed causal updates could be completed and all timers could return to zero).

4. DISCUSSION

Storage costs associated to synaptic weight memory arrangements have been previously studied. In Moradi et al. (2013), the authors describe a network clustering scheme which uses a two-stage routing architecture to reduce the overall memory storage requirements. This method is also mentioned in Joshi et al. (2017) and is referred to as "clustered addressing." In both of these studies, the storage savings comes at the cost of reduced flexibility in network connectivity, since a specific topology must exist for groups of neurons to be clustered together. Instead, we decided not to constrain our networks to any structured topology. In Joshi et al. (2017), the authors describe the data structures we have presented, highlighting, particularly, the storage cost savings obtained for a large range of connectivity density when using the PB-BMP architecture. However, the impact of pointer-based models on learning algorithms was only briefly mentioned, and memory access costs were not analyzed. More recently, the impact of using different memory arrangements on spike routing and network traffic congestion was described in Kornijcuk et al. (2018). Though the work describes a theoretical means of routing-rate evaluation and results for maximum network sizes for each of their memory arrangements, it does not target any specific learning algorithm, and the experimental results focus only on an inference task without synaptic plasticity. More recently, the authors in Kim et al. (2018) proposed a modified SRAM which enables transposable memory access. The method is interesting as it facilitates the reverse (post-to-pre) access for causal updates; however, it can only be applied to fully connected network topologies (i.e., crossbars), and, thus, are not efficient for representing sparse networks since compressed data structures are typically not transposable.

In terms of spike-driven learning, there have been multiple attempts to replicate or approximate STDP with forward-only connectivity. The motivation for storing synaptic weights in a pre-synaptic perspective (i.e., pre-to-post) is because post-synaptic-driven systems are not as efficient in terms of number of memory accesses as pre-synaptic-driven systems; this is mainly because, as we sweep through neurons to update their states during a system time step, Δt, for each post-synaptic neuron we must verify the spike state of every pre-synaptic neuron, even if none of these has spiked. Conversely, pre-synaptic-driven systems operate in an on-demand fashion, accessing the pre-synaptic spike states only as needed.

In Pedroni et al. (2016); Detorakis et al. (2018), we described a less-detailed version of our method; yet, we did not study all the data structures nor were we able to address all of the drawbacks incurred by delayed causal updates (as we have shown in the current paper). One of the earliest works which evaluated the complexity of implementing the STDP learning algorithm in a neuron address domain was presented in Vogelstein et al. (2003). The authors discussed how the address-event representation (AER) protocol could support STDP learning in the address domain. Being pioneering work, the paper considered only small networks, consequently not addressing the different possible arrangements for organizing synaptic weights in memory and the implications of requiring reverse access for performing causal updates.

Methods that approximate STDP learning by equally updating all the synaptic weights based on recent spike activity have been proposed. In Bichler et al. (2012), the authors use a special form of STDP which equally depresses all the synapses that did not recently contribute to the post-synaptic spike activation regardless of their activation time; in contrast, synapses that were activated with a pre-synaptic spike a short time before post-synaptic spikes are strongly potentiated. The authors in Yousefzadeh et al. (2017) created a more hardware-friendly version of this model by limiting the number of synapses to be potentiated (instead of limiting the STDP time window duration), eliminating the need for time-stamping the spikes. Though efficient in terms of memory access, with both of these methods it is not possible to depress synapses whose activation time is precisely not correlated with the post-synaptic spike, and the methods only work if LTD is systematically applied to synapses not undergoing an LTP. Additionally, the methods are post-synaptic-driven, undergoing the aforementioned drawbacks of this mechanism.

Another alternative to approximating STDP is by using other variables (usually post-synaptic membrane potential) as a proxy for the post-synaptic spike times when computing causal updates. This learning rule was proposed in Brader et al. (2007) and has even been incorporated in the SpiNNaker system (Davies et al., 2012; Lagorce et al., 2015). More recent work describes how to use the rule for learning sequences of spikes (Sheik et al., 2016). Once again, though very efficient in terms of memory access and spike time storage, in this method exact STDP is not possible as post-synaptic potential serves only as a [deterministic (Lagorce et al., 2015) or probabilistic (Sheik et al., 2016)] proxy of the post-synaptic spike time and, in many cases, is not capable of capturing the subtle spike time causalities of STDP.

The third category of methods for approximating STDP consists on delaying the weight updates, and is the category which our proposed method falls under. In the Loihi system, the authors adopt a less event-driven method where synaptic modification is performed in an epoch-based mechanism (Davies et al., 2018). Their method delays the updating of all synaptic states to the end of a periodic learning epoch time, and, to avoid receiving more than one spike in a given epoch, the epoch period is normally set to the minimum refractory delay of all neurons in the network. Though Loihi implements forward connectivity tables for supporting generalized STDP rules, the periodic servicing (i.e., non-event-driven methodology) can result in inexact weights being delivered to post-synaptic neurons since multiple pre-synaptic spikes may occur before a weight update takes place. Therefore, certain conditions in firing rates must be guaranteed for their method to be equivalent to STDP.

In the current version of the SpiNNaker system, STDP learning is approximated using a trace-based approach via delayed updates (Mikaitis et al., 2018). Since in trace-based STDP each spike leaves an exponentially decaying trace (Morrison et al., 2008), this renders possible linearly accumulating the spike traces into a single variable, representing the total current effect of all past spikes. In this manner, weight updates can then be performed in an online fashion at the onset of either pre- or post-synaptic spikes. In SpiNNaker, however, the updates only occur at the onset of pre-synaptic spikes, meaning that, for the method to follow rather closely to original STDP, the system relies on frequently firing pre-synaptic neurons. This issue can be observed in the case when a post-synaptic neuron spikes multiple times soon after a pre-synaptic spike (typically resulting in large causal updates): if the pre-synaptic neuron spikes again in a much later time, then the causal updates will be practically null due to the almost completely decayed traces (somewhere along the lines of the problem encountered in case 4 in **Figure 4D**). Additionally, besides serving only as an approximation to STDP, the trace-based method requires an exponentially decaying kernel, and, thus, other kernels such as those in **Figure 1C** cannot be implemented.

Perhaps the most similar work to ours has been presented in Jin et al. (2010), which uses a deferred-event approach and stores spike times for postponed processing at the time of the next event following them. This method has been previously implemented in the SpiNNaker system under their "deferred event driven model" (Rast et al., 2008; Diehl and Cook, 2014; Galluppi et al., 2015). It is similar to our proposed method in that weight updates are driven by pre-synaptic spikes and causal updates are delayed; however, some important distinctions should be highlighted:

- A neuron's spike history is stored as a bitmap in an array. However, as presented in **Appendix A1**, using multiple timers is at least as efficient as using a bitmap array, and becomes extremely more efficient for large T_{refr}.
- Acausal updates are not immediately processed and are also deferred to the future, once more pre-synaptic spikes have arrived. This implies that larger arrays are required to store spikes on both sides of the STDP window for post-synaptic neurons. In fact, in their work the post-synaptic bitmap array is three times larger than the pre-synaptic array. In our solution, applying the acausal updates immediately at

the onset of pre-synaptic spikes demands that we use timers that must cover only one side (i.e., the longest side) of the STDP window.

- Since the bitmap array is only updated at the onset of new spikes (but not necessarily at the expiration of the pre-synaptic STDP timer) and STDP updates can only take place when an "old" pre-synaptic spike eventually exits the bitmap array, this means that *both* causal and acausal updates rely on frequently firing pre-synaptic neurons. This demands that pre-synaptic spikes arrive at a high enough rate to ensure that the pre-synaptic spike time bitmap array is frequently updated so weight updates are not lost. In their work, the minimum firing rate for pre-synaptic neurons is 10.4 Hz.
- Since multiple pre-synaptic spikes may occur before an "old" pre-synaptic spike eventually exits the bitmap array, this implies that the weights being used for updating post-synaptic neuron variables at each pre-synaptic spike event could (or most likely will) be an "old" set of weights since the causal and acausal updates have been deferred. Therefore, though qualitatively similar, a quantitative equivalence with the original STDP algorithm will probably not occur.

5. CONCLUSIONS

There are multiple forms of organizing data structures for storing synaptic weights. Among these different memory arrangements, pointer-based models are capable of data compression by storing only the existent connections in the network. In pointer-based models, weights are stored, in a high-level sense, as lists of post-synaptic addresses and weights, where the pointer to the list is defined by the pre-synaptic neuron address. Biologically relevant neural networks are typically unstructured and sparsely connected, making pointer-based architectures particularly efficient at storing these network topologies. In this work, we studied the storage costs (in bits) of each data structure and identified the most efficient based on network parameters (e.g., network size and weight bit-length) and connectivity density.

For the different data structures, we analyzed the computational complexity (in number of memory accesses) of obtaining synaptic address and weight when accessing the tables in forward and reverse directions. Though efficient in terms of storage for a wide range of connectivity density values, pointer-based models natively present only forward connectivity access, making them inefficient when implementing spike-time-based local learning rules such as STDP—which requires both forward (pre-to-post) and reverse (post-to-pre) connectivity access. Therefore, we devised a novel means of efficiently implementing STDP by forward-only synaptic connectivity access, benefiting from the reduced memory storage property of pointer-based data structures. In the traditional STDP algorithm, causal updates are performed at the onset of post-synaptic spikes, demanding reverse access at this instant. Our proposed method operates by delaying the causal weight updates until the instant of expiration of the pre-synaptic STDP timer. With this, forward access is performed for both causal and acausal updates, driven by pre-synaptic events.

Natural drawbacks arise when delaying the causal updates, particularly with respect to high-firing post-synaptic neurons. All the drawbacks can be addressed by a very simple rule: the number of STDP timers for each neuron should be equal to the number of spikes which can occur inside the STDP learning window. This rule can be obtained by using multiple timers when $T_{refr} < T_{stdp}$, with each timer lasting T_{refr} time steps. Using this strategy results in the possibility of implementing nearest-neighbor and all-to-all temporal spike interaction. Additionally, by extending the number of timers, the more complex triplet-based temporal interaction can also be deployed.

Lastly, besides the comparison of storage and access costs and efficiencies for each data structure, we devised a budget efficiency figure of merit for a trade-off analysis of the benefits of each model depending on application requirements and storage and access budget. In sum, we feel our work is unique in that it presents a methodology for identifying the optimal memory arrangement solution based on system requirements and network topology, including also the cost of memory access, and supplying the first viable and exact solution for implementing STDP learning in systems organized with either crossbar arrays or forward-only connectivity tables.

AUTHOR CONTRIBUTIONS

BP and GC developed the main part of the work, including the algorithms, simulations, analyses, and results. All authors contributed to the manuscript.

FUNDING

This work was partly supported by the National Science Foundation (CNS-1823366), the Office of Naval Research (N00014-18-1-2248), the Brazilian National Council of Technological and Scientific Development (CNPq-CsF 201174/2012-0), and Intel Corporation.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2019.00357/full#supplementary-material

REFERENCES

Andreou, A. G., Meitzler, R. C., Strohbehn, K., and Boahen, K. (1995). Analog VLSI neuromorphic image acquisition and pre-processing systems. *Neural Netw.* 8, 1323–1347. doi: 10.1016/0893-6080(95)00098-4

Bassett, D. S., and Bullmore, E. (2006). Small-world brain networks. *Neuroscientist* 12, 512–523. doi: 10.1177/1073858406293182

Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J.-P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Netw.* 32, 339–348. doi: 10.1016/j.neunet.2012.02.022

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* 19, 2881–2912. doi: 10.1162/neco.2007.19.11.2881

Bullmore, E., and Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and functional systems. *Nat. Rev. Neurosci.* 10:186. doi: 10.1038/nrn2575

Caporale, N., and Dan, Y. (2008). Spike timing-dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.* 31, 25–46. doi: 10.1146/annurev.neuro.31.060407.125639

Dan, Y., and Poo, M.-m. (2004). Spike timing-dependent plasticity of neural circuits. *Neuron* 44, 23–30. doi: 10.1016/j.neuron.2004.09.007

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Davies, S., Galluppi, F., Rast, A. D., and Furber, S. B. (2012). A forecast-based STDP rule suitable for neuromorphic implementation. *Neural Netw.* 32, 3–14. doi: 10.1016/j.neunet.2012.02.018

Detorakis, G., Sadique Sheik, C. A., Paul, S., Pedroni, B. U., Dutt, N., Krichmar, J., et al. (2018). Neural and synaptic array transceiver: a brain-inspired computing framework for embedded learning. *Front. Neurosci.* 12:583. doi: 10.3389/fnins.2018.00583

Diehl, P. U., and Cook, M. (2014). "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware," in *IJCNN* (Beijing), 4288–4295.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.1109/IJCNN.2014.6889876

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Galluppi, F., Lagorce, X., Stromatias, E., Pfeiffer, M., Plana, L. A., Furber, S. B., et al. (2015). A framework for plasticity implementation on the SpiNNaker neural architecture. *Front. Neurosci.* 8:429. doi: 10.3389/fnins.2014.00429

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511815706

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). "Generative adversarial nets," in *Advances in Neural Information Processing Systems* (Montréal), 2672–2680.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv:1308.0850*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (Vancouver: IEEE), 6645–6649.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.1109/ICASSP.2013.6638947

Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cereb. Cortex* 17, 2443–2452. Ioffe and Szegedy, 2015

Jin, X., Rast, A., Galluppi, F., Davies, S., and Furber, S. (2010). "Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware," in *Neural Networks (IJCNN), The 2010 International Joint Conference on* (Barcelona: IEEE), 1–8.

Joshi, S., Pedroni, B. U., and Cauwenberghs, G. (2017). "Neuromorphic event-driven multi-scale synaptic connectivity and plasticity," in *Signals, Systems, and Computers, 2017 51st Asilomar Conference on* (Pacific Grove, CA: IEEE), 1–5.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1109/IJCNN.2010.5596372

Kim, J., Koo, J., Kim, T., and Kim, J.-J. (2018). Efficient synapse memory structure for reconfigurable digital neuromorphic hardware. *Front. Neurosci.* 12:829. doi: 10.3389/fnins.2018.00829

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv:1412.6980*.

Koch, C., and Mathur, B. (1996). Neuromorphic vision chips. *IEEE Spectrum* 33, 38–46.

Kornijcuk, V., Park, J., Kim, G., Kim, D., Kim, I., Kim, J., et al. (2018). Reconfigurable spike routing architectures for on-chip local learning in neuromorphic systems. *Adv. Mater. Technol.* 4:1800345. doi: 10.1002/admt.201800345

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (Lake Tahoe), 1097–1105.

Lagorce, X., Stromatias, E., Galluppi, F., Plana, L. A., Liu, S.-C., Furber, S. B., et al. (2015). Breaking the millisecond barrier on SpiNNaker: implementing asynchronous event-based plastic models with microsecond resolution. *Front. Neurosci.* 9:206. doi: 10.3389/fnins.2015.00206

Liu, S.-C., and Delbruck, T. (2010). Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* 20, 288–295. doi: 10.1016/j.conb.2010.03.007

Maher, M. A. C., Deweerth, S. P., Mahowald, M. A., and Mead, C. A. (1989). Implementing neural architectures using analog VLSI circuits. *IEEE Trans. Circ. Syst.* 36, 643–652.

Mahowald, M. A. (1993). *The Address-Event Representation Communication Protocol. AER 0.02*. Pasadena, CA: California Institute of Technology.

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213–215.

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mikaitis, M., Pineda García, G., Knight, J. C., and Furber, S. B. (2018). Neuromodulated synaptic plasticity on the SpiNNaker neuromorphic system. *Front. Neurosci.* 12:105. doi: 10.3389/fnins.2018.00105

Moradi, S., Imam, N., Manohar, R., and Indiveri, G. (2013). "A memory-efficient routing method for large-scale spiking neural networks," in *Circuit Theory and Design (ECCTD), 2013 European Conference on* (Dresden: IEEE), 1–4. doi: 10.1109/ECCTD.2013.6662203

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybern.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Nair, V., and Hinton, G. E. (2010). "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa), 807–814.

Nessler, B., Pfeiffer, M., and Maass, W. (2009). "STDP enables spiking neurons to detect hidden causes of their inputs," in *Advances in Neural Information Processing Systems* (Vancouver), 1357–1365.

Oliver, B. (1952). Efficient coding. *Bell Syst. Tech. J.* 31, 724–750.

Park, J., Yu, T., Joshi, S., Maier, C., and Cauwenberghs, G. (2017). Hierarchical address event routing for reconfigurable large-scale neuromorphic systems. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 2408–2422. doi: 10.1109/TNNLS.2016.2572164

Pedroni, B. U., Sheik, S., Joshi, S., Detorakis, G., Paul, S., Augustine, C., et al. (2016). "Forward table-based presynaptic event-triggered spike-timing-dependent plasticity," in *Biomedical Circuits and Systems Conference (BioCAS)* (Shanghai: IEEE), 580–583.

Pfister, J.-P., and Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *J. Neurosci.* 26, 9673–9682. doi: 10.1523/JNEUROSCI.1425-06.2006

Rast, A., Jin, X., Khan, M., and Furber, S. (2008). "The deferred event model for hardware-oriented spiking neural networks," in *International Conference on Neural Information Processing* (Berlin: Springer), 1057–1064.

Rossum, M. v. (2001). A novel spike distance. *Neural Comput.* 13, 751–763. doi: 10.1162/089976601300014321

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323:533. doi: 10.1038/323533a0

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*, Vol. 82. Philadelphia, PA: SIAM. doi: 10.1137/1.9780898718003

Seeman, S. C., Campagnola, L., Davoudian, P. A., Hoggarth, A., Hage, T. A., Bosma-Moody, A., et al. (2018). Sparse recurrent excitatory connectivity in the microcircuit of the adult mouse and human cortex. *bioRxiv* 292706. doi: 10.7554/eLife.37349

Sheik, S., Paul, S., Augustine, C., and Cauwenberghs, G. (2016). "Membrane-dependent neuromorphic learning rule for unsupervised spike pattern detection," in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Shanghai: IEEE), 164–167.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484. doi: 10.1038/nature16961

Sjöström, J., and Gerstner, W. (2010). Spike-timing dependent plasticity. *Front. E-books* 35.

Song, S., and Abbott, L. F. (2001). Cortical development and remapping through spike timing-dependent plasticity. *Neuron* 32, 339–350. doi: 10.1016/S0896-6273(01)00451-2

Vogelstein, R. J., Tenore, F., Philipp, R., Adlerstein, M. S., Goldberg, D. H., and Cauwenberghs, G. (2003). "Spike timing-dependent plasticity in the address domain," in *Advances in Neural Information Processing Systems* (Vancouver), 1171–1178.

Yousefzadeh, A., Masquelier, T., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017). "Hardware implementation of convolutional STDP for on-line visual feature learning," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore: IEEE), 1–4.

Conflict of Interest Statement: SP and CA were employed by company Intel Corporation. SS was employed by company aiCTX.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 26 April 2019
doi: 10.3389/fnins.2019.00405

A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications

*Yuhan Shi, Leon Nguyen, Sangheon Oh, Xin Liu and Duygu Kuzum**

Electrical and Computer Engineering Department, University of California, San Diego, San Diego, CA, United States

Inspired from the computational efficiency of the biological brain, spiking neural networks (SNNs) emulate biological neural networks, neural codes, dynamics, and circuitry. SNNs show great potential for the implementation of unsupervised learning using in-memory computing. Here, we report an algorithmic optimization that improves energy efficiency of online learning with SNNs on emerging non-volatile memory (eNVM) devices. We develop a pruning method for SNNs by exploiting the output firing characteristics of neurons. Our pruning method can be applied during network training, which is different from previous approaches in the literature that employ pruning on already-trained networks. This approach prevents unnecessary updates of network parameters during training. This algorithmic optimization can complement the energy efficiency of eNVM technology, which offers a unique in-memory computing platform for the parallelization of neural network operations. Our SNN maintains ∼90% classification accuracy on the MNIST dataset with up to ∼75% pruning, significantly reducing the number of weight updates. The SNN and pruning scheme developed in this work can pave the way toward applications of eNVM based neuro-inspired systems for energy efficient online learning in low power applications.

Keywords: spiking neural networks, unsupervised learning, handwriting recognition, pruning, in-memory computing, emerging non-volatile memory

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Richard Miru George,
Dresden University of Technology,
Germany
Priyadarshini Panda,
Purdue University, United States

****Correspondence:***
Duygu Kuzum
dkuzum@eng.ucsd.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *23 November 2018*
Accepted: *09 April 2019*
Published: *26 April 2019*

Citation:
Shi Y, Nguyen L, Oh S, Liu X and Kuzum D (2019) A Soft-Pruning Method Applied During Training of Spiking Neural Networks for In-memory Computing Applications. Front. Neurosci. 13:405. doi: 10.3389/fnins.2019.00405

INTRODUCTION

In recent years, brain-inspired spiking neural networks (SNNs) have been attracting significant attention due to their computational advantages. SNNs allow sparse and event-driven parameter updates during network training (Maass, 1997; Nessler et al., 2013; Tavanaei et al., 2016; Kulkarni and Rajendran, 2018). This results in lower energy consumption, which is appealing for hardware implementations (Cruz-Albrecht et al., 2012; Merolla et al., 2014; Neftci et al., 2014; Cao et al., 2015). Emerging non-volatile memory (eNVM) arrays have been proposed as a promising in-memory computing platform to implement SNN training in an energy efficient manner. eNVM devices can implement spike-timing-dependent plasticity (STDP) (Jo et al., 2010; Kuzum et al., 2011), which is a commonly used weight update rule in SNNs. Most demonstrations utilize eNVM crossbar arrays to parallelize computation of the inner product (Alibart et al., 2013; Choi et al., 2015; Prezioso et al., 2015; Eryilmaz et al., 2016; Ge et al., 2017; Wong, 2018). In addition, there are several works focus on using eNVM hardware such as spintronic devices or crossbars with additional algorithmic optimization of STDP learning rules to perform hardware implementation of SNN (Sengupta et al., 2016; Srinivasan et al., 2016; Ankit et al., 2017; Panda et al., 2017a,b). While eNVM crossbar arrays improve energy efficiency at a device level for SNN training, network level algorithmic optimization is still important to further improve energy efficiency for wide adoption of SNNs in low power applications.

Pruning network parameters, i.e., synaptic weights, is a recent algorithmic optimization (Han et al., 2015) that is widely used for compressing the network to improve the energy efficiency for the inference operation of deep neural networks. Although synaptic pruning has been demonstrated in many biophysical SNN models (Iglesias and Villa, 2007; Deger et al., 2012, 2017; Kappel et al., 2015; Spiess et al., 2016), how the pruning can be used for non-biophysical SNN has not been fully explored yet. Moreover, this method is applied on already-trained networks and it does not address the high-energy consumption during training, which requires iterative weight updates. A new approach toward network training that improves the energy efficiency of SNNs is crucial to develop online learning systems that can learn and perform inference in real world scenarios.

Here, we develop an algorithm to prune during training for SNNs with eNVMs to improve network level energy efficiency for in-memory computing applications. Although Rathi et al. (Rathi et al., 2018) has showed pruning in SNN before, there are several key innovations and differences of the pruning method in this work compared to Rathi et al.' work. Our method considers the spiking activity of the output neurons to decide when to prune during the training while Rathi et al. performs the pruning at regular intervals for every batch without considering the characteristics of the output neurons. In addition, once the weights have been pruned during the training, we do not update the pruned weights for the rest of the training while Rathi et al. only temporally removes the pruned weights and they can still be updated when new batches present to the network. Finally, we develop soft-pruning as an extension of pruning. Soft-pruning sets the pruned weights to a constant non-zero values. Therefore, it is novel in terms of treating pruned weights. Rathi et al. only implement pruning.

Our paper is organized as follows: first, we describe our unsupervised SNN model and the weight update rule. Then, we introduce a pruning method that exploits spiking characteristics of the SNN to decrease the number of weight updates and thus energy consumption during training. Finally, we discuss how our SNN training and pruning algorithm can potentially be realized using eNVM crossbar arrays and perform circuit-level simulations to confirm the feasibility for online unsupervised learning to reduce the energy consumption and training time.

In section Input layer to section Testing, we discuss our SNN model and the algorithms relating to weight updates. In section Pruning during training, we discuss methods to prune during training. In section Results and discussion, we discuss our software simulation results, compare our SNN with state-of-the-art unsupervised SNN algorithms on MNIST and explore the method to implement our SNN model and pruning algorithm using the eNVM crossbar array through circuit-level simulations.

NEURAL NETWORK ARCHITECTURE

Inspired by the information transfer in biological neurons via precise spike timing, SNNs temporally encode the inputs and outputs of a neural network layer using spike trains. The weights of the SNN are updated via a biologically plausible STDP, which modulates weights based on the timing of input and output spikes (Nessler et al., 2013; Tavanaei et al., 2016). This can be easily implemented on an eNVM crossbar array (Kuzum et al., 2011), making it ideal for online learning in hardware.

Our SNN performs unsupervised classification of handwritten digits from the MNIST dataset. It is a single layer network defined by the number of inputs neurons *n*, the number of outputs neurons *m*, and an *m* by *n* weight matrix. The number of input neurons can vary depending on preprocessing, but by default there are 784 input neurons to account for each grayscale pixel in a training sample. The output layer consists of 500 neurons to classify the 10 classes of the MNIST dataset (60,000 training images and 10,000 testing images). **Figure 1** describes the fully connected network architecture.

As an overview of the pipeline, we first train the SNN by sequentially presenting samples from the training set. The purpose of training is to develop the weights of each output neuron so that they selectively fire for a certain class in MNIST. Afterwards, we present the training set for a second time to label each trained output neuron with the class of training samples that has the highest mean firing rate. This organizes the output neurons into populations that each respond to one of the classes. Finally, we test the SNN by predicting the label of each of the test samples based the class of output neurons with the highest mean firing rate.

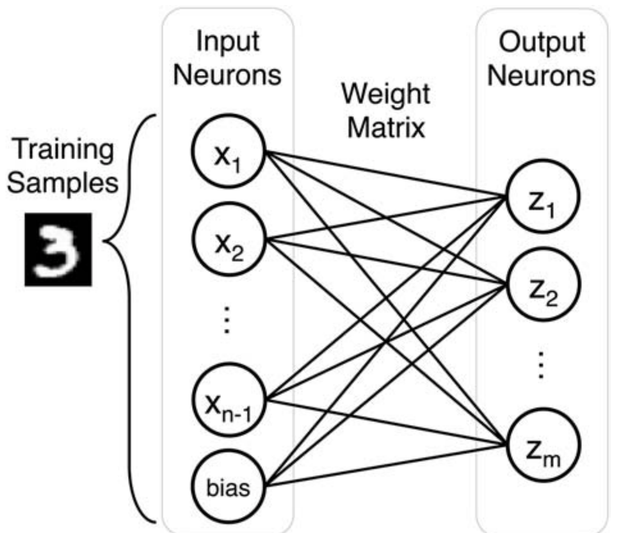


FIGURE 1 | Each training sample is represented by *n* input neurons (*n*-1 input neurons and a bias term). The weight matrix is an *m* by *n* array composed of the weights that connect input neurons to output neurons in the single layer network.

Input Layer

We first remove the pixels that are used to represent the background in at least 95% of the training samples to reduce the number of input layer neurons. Because the grayscale pixels have intensity values in the range [0, 1], the pixels with a value of 0 correspond to the background and are thus checked for removal. After this step, we retain 397 of the original 784 pixels, reducing the complexity of the SNN. Therefore, we have 398 input neurons for a given training sample after accounting for an additional bias input neuron, which has a value of 1. Our output neurons do not have refractory periods and there is no lateral inhibition between them.

We encode each of these inputs as a Poisson spike train at a frequency of 200 times its value, leading to a maximum input firing rate of 200 Hz. We round the timing of each spike that is generated by the Poisson process to the nearest millisecond, which is the time of one time step in the SNN. The SNN displays each training sample for the first 40 ms of a 50 ms presentation period, and thus the input spikes for a given training sample can only occur in this 40 ms window. **Figure 2A** shows an example of the input spiking activity for the duration of three training samples.

Output Layer

For output spikes, we use the Bayesian winner-take-all (WTA) firing model (Nessler et al., 2013). Unlike traditional integrate-and-fire models (Gupta and Long, 2007; Diehl and Cook, 2015a), this model is shown to demonstrate Bayes' rule (Nessler et al., 2013), which is a probabilistic model for learning and cognitive development (Perfors et al., 2011). The SNN fires an output spike from any given output neuron according to a 200 Hz Poisson process. The output neuron that fires is chosen from a softmax distribution of the output neurons' membrane potentials:

$$\mathrm{p}\left(u_k\right) = \frac{\exp\left(u_k\right)}{\sum_{i=1}^{m} \exp\left(u_i\right)}, \tag{1}$$

where $\left\{\mathrm{p}\left(u_k\right)\right\}_{k=1,\,\ldots,m}$ is the softmax probability distribution of the membrane potentials $\{u_k\}_{k=1,\,\ldots,m}$. m is the number of output neurons. Our firing mechanism is probabilistic instead of hard thresholding the membrane potentials. Therefore, the neuron with higher membrane potential means that it has higher chance to fire. We calculate membrane potentials u_k using (2)

$$u_k = \sum_i W_{ki} X_i + b_k \tag{2}$$

where W_{ki} is the weight between input neuron i and output neuron k, X_i is the spike train generated by input neuron i and b_k is the weight of the bias term. Equation (2) calculates an output neuron's membrane potential as the inner product between the input spikes at a given time step and the output neuron's weights, but this does not need to be integrated with each time step. Instead, we only calculate the membrane potentials at time steps when an output neuron fires because it is only used to determine which output neuron to fire. This removes additional parameters and resources needed with typical integrate-and-fire neuron models, which use the membrane potential to also find when to fire output neurons, allowing for a more efficient hardware implementation.

Weight Updates: STDP Rule

When an output neuron fires, a simple STDP rule determines which weights to update via long-term potentiation (LTP) or long-term depression (LTD). As shown in **Figure 3A**, if an input neuron's most recent spike is within $\sigma = 10$ ms of the output spike, then the weight for this input-output synapse is increased (LTP). Otherwise, if it is beyond this 10 ms window of the output spike, then the weight is decreased (LTD).

This 10 ms window is in accordance with the fact that training samples are not displayed during the final 10 ms of their presentation period—they are only displayed for the first 40 ms of the 50 ms presentation period. Thus, there are no input spikes in the final 10 ms of each presentation, as seen in **Figure 2A**. Therefore, this STDP window prevents LTP weight updates that are potentially caused by the input spiking activity of the previous training sample. For example, when a new training sample is inputted to the SNN, an output spike occurring at simulation time $t = 50$ ms cannot have a spike-timing difference with an input spike occurring from $t = 41$ ms to $t = 49$ ms, since this is within the 10 ms window for LTP weight updates.

Figure 2B shows an example of the output spiking activity for 10 representative output neurons with randomly initialized weights, illustrating the random spiking activity of an untrained SNN. The effect of performing weight updates is to train the network to selectively fire to certain classes of inputs. At the start of training, we randomly initialize all weight values between [−1, 1], and the LTP and LTD update rules keep the weight values within the range [−1, 1]. The LTP weight update is an exponential function of the form $\Delta w_{LTP}(w) = ae^{-b(w+1)}$ (**Figure 3B**), where $a \in \{\mathbb{R} : 0 < a < 1\}$ and $b \in \mathbb{R}_{>0}$ are parameters that control the scale of the exponential, and w is the current weight value. For LTP updates to keep weight values within the upper bound of 1, we pick the parameters such that the weight update decays toward 0 as the current weight approaches 1. As a result, exponential LTP updates will guarantee that the weights converge to the upper bound of 1.

Unlike LTP, the LTD weight update is a constant function that disregards the current weight value: $w_{LTD} = -c$, where $c \in \{\mathbb{R} : 0 < c < 1\}$ is a parameter that controls the magnitude of the weight decrease. Because there is no guarantee of convergence as with the exponential LTP update, the SNN clips weights to the lower bound of −1. Alternatively, we can have an exponential LTD update that is mirrored about $w = 0$ from the exponential LTP update, i.e., $\Delta w_{LTD}(w) = -ae^{b(w-1)}$, and choose parameters to have weight convergence as in the case of LTP. However, the constant LTD update is easier to implement in hardware since there are less parameters to tune. The specific parameter choices of a, b ,and c are shown in **Table 1** and they come from cross validation of the parameter set to optimize the classification accuracy. Several previously published papers have proposed probabilistic synapses to perform STDP weight update (Vincent et al., 2014; Srinivasan et al., 2016). It is worth to note that the synapses in our network is deterministic and only the firing mechanism

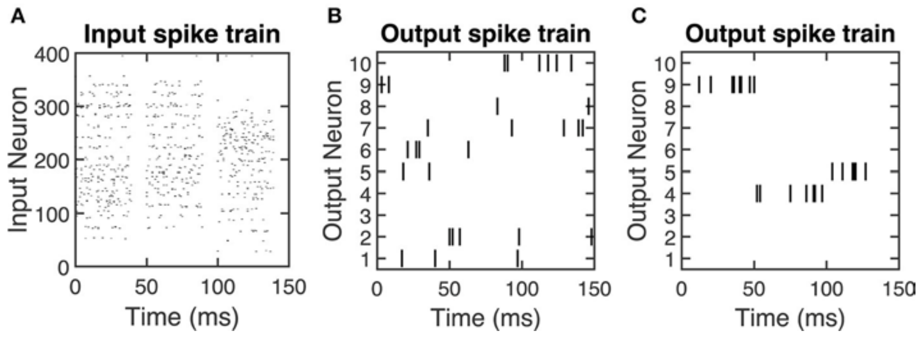


FIGURE 2 | Spike raster plots showing examples of **(A)** input spiking activity, **(B)** output spiking activity for an untrained SNN, and **(C)** output spiking activity for a trained SNN. For **(B)** and **(C)**, 10 output neurons' spiking activities are selected as a representative example. After the SNN is trained, the output spike firing activity is more coordinated, which is indicated by the output neurons selectively firing to certain input stimuli. The time duration on the *x* axis indicates the presentation of training samples. Since the output neuron firing rate is 200 Hz, therefore there are around 10 spikes (# of spikes = presentation time × frequency = 50ms × 0.001 × 200 Hz = 10) will be generated within 50 ms presentation time.

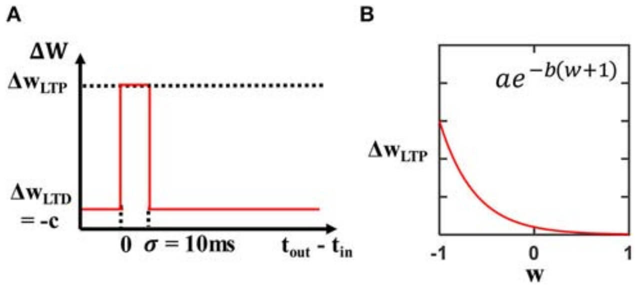


FIGURE 3 | (A) STDP rule showing the 10 ms window for an output-input spike time difference (t_{out}–t_{in}) that determines whether an LTP or an LTD update is performed. If the output-input spike time difference (t_{out}–t_{in}) is within 10 ms, the weight corresponding to this input-output synapse is updated via LTP. Otherwise, the weight is updated via LTD. The LTP update is an exponential function that depends on the current weight, and the LTD update is a constant. **(B)** The exponential LTP update is dependent on the current weight *w* and it helps keep the weight values within the range [−1, 1].

TABLE 1 | Simulation parameters used in training, labeling and testing for this work.

Parameters		10–digits		
		Training	Labeling	Testing
# of neuron	Input		398	
	Output		500	
Firing rate (Hz)	Input	200	200	200
	Output	200	200	600
Image presenting time (ms)		50	50	200
Neuron removal threshold		–	0.75	–
Pruning threshold	Prune parameter *(r)*	10	–	–
	Spice count	8	–	–

STDP		a = 0.0667
		b = 2.5
		c = 0.0167		

of output neurons is probabilistic as explained in section Output layer.

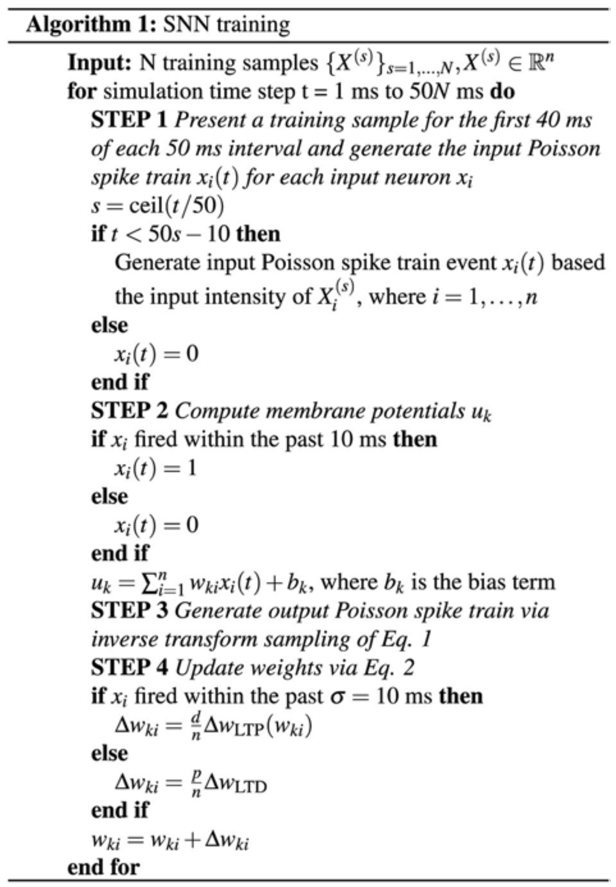
Scaling Weight Updates as a Normalization Method

To perform a weight update, we add to the current weight w_t the weight update, which is scaled by an additional factor depending on whether the update is LTP or LTD:

$$w_{t+1} = \begin{cases} w_t + \frac{d}{n}\Delta w_{LTP}(w_t), & LTP \\ w_t + \frac{p}{n}\Delta w_{LTD}, & LTD \end{cases} \tag{3}$$

where d is the number of weights to undergo LTD, p is the number of weights to undergo LTP, and n is the total number of weights for an output neuron, which also corresponds to the number of input neurons. Because of the STDP rule, all n weights of an output neuron are updated at any given output neuron firing event, which means that $d + p = n$. Because the number of LTP updates is often disproportionate with that of LTD due to the probabilistic spike firing, the scaling factors d and p keep the net weight change of both types of updates proportional so that for all output neurons, the distribution of weight values have roughly the same mean and variance. With this, an overview of the SNN training method is outlined in **Figure 4**.

This scaling of LTP and LTD weight updates is used to prevent certain output neurons from firing more than others. It effectively normalizes the weight distributions of each output neuron so that they fire according to the correlation between their weights and the training sample, rather than firing because the magnitude of their weights artificially increases their membrane potential. This foregoes the need to normalize the weight distributions of each output neuron through calculating the mean and standard deviation, which requires additional resources when implementing the weight update in hardware.



```
Algorithm 1: SNN training
    Input: N training samples ${X^{(s)}}_{s=1,...,N}, X^{(s)} \in \mathbb{R}^n$
    for simulation time step t = 1 ms to 50N ms do
        STEP 1 Present a training sample for the first 40 ms
        of each 50 ms interval and generate the input Poisson
        spike train $x_i(t)$ for each input neuron $x_i$
        $s = \mathrm{ceil}(t/50)$
        if $t < 50s - 10$ then
            Generate input Poisson spike train event $x_i(t)$ based
            the input intensity of $X_i^{(s)}$, where $i = 1, \ldots, n$
        else
            $x_i(t) = 0$
        end if
        STEP 2 Compute membrane potentials $u_k$
        if $x_i$ fired within the past 10 ms then
            $x_i(t) = 1$
        else
            $x_i(t) = 0$
        end if
        $u_k = \sum_{i=1}^{n} w_{ki} x_i(t) + b_k$, where $b_k$ is the bias term
        STEP 3 Generate output Poisson spike train via
        inverse transform sampling of Eq. 1
        STEP 4 Update weights via Eq. 2
        if $x_i$ fired within the past $\sigma = 10$ ms then
            $\Delta w_{ki} = \frac{d}{n} \Delta w_{\mathrm{LTP}}(w_{ki})$
        else
            $\Delta w_{ki} = \frac{p}{n} \Delta w_{\mathrm{LTD}}$
        end if
        $w_{ki} = w_{ki} + \Delta w_{ki}$
    end for
```

FIGURE 4 | SNN training algorithm.

Testing

After training is done, we fix the trained weights and assign a class to each neuron by the following steps: First, we present the whole training set to the SNN and record the cumulative number of output spikes N_{kj}, where $k = 1, ..., m$ (m is number of output neurons) and $j = 1, \ldots, n$ (n is number of classes, for MNIST, $n = 10$). Then, for each output neuron i, we calculate its response probability Z_{kj} to each class j using Eq. (4). Finally, each neuron k is assigned to the class that gives the highest response probability Z_{kj}.

$$Z_{kj} = \frac{N_{kj}}{\sum_{j=1}^{n} N_{kj}} \tag{4}$$

After training and labeling are done, we fix the weights and present test set to our network. We use Eq. (5) to predict the class of each sample, where S_{jk} is the number of spikes for the kth output neuron that are labeled as class j and N_j is the number of output neurons labeled as class j.

$$J = \underset{j}{argmax} \frac{\sum_{k=1}^{N_j} S_{jk}}{N_j} \tag{5}$$

Pruning During Training

Pruning is a concept in machine learning that removes redundant branches from a decision tree to reduce complexity and improve accuracy of the classifier. It prevents overfitting by learning the general structure of the input data instead of learning minute details. Han et al. implement pruning on trained convolutional neural networks to remove unimportant weights that have low contribution to the output (Han et al., 2015). For example, weights with values close to 0 can be removed since their inner product with their respective inputs will yield low output values. This removal effectively sets the weight values to 0, allowing for a sparser representation of the network for mobile applications while still retaining the same classification performance. Instead of pruning after training, we propose a method to prune during training on SNNs to reduce the number of weight updates.

Our implementation of pruning removes unimportant weights belonging to each output neuron, and each output neuron is only pruned once during training. When an output neuron fires, its weights can potentially be pruned based on the level of development in its weights. There is a tradeoff in choosing when to prune an output neuron. If we prune weights early during training, we save computation by not having to update these weights later on. However, by pruning early, the weights might not be trained enough to recognize a certain class in the dataset at the time of pruning, and this early pruning can hamper the future development of the weights. Conversely, pruning late better insures that the weights are trained at the expense of computing more weight updates.

To determine when to prune the weights of an output neuron, we refer to the spiking activity of the output neurons. The output neuron spiking activity is an inherent feature of SNNs that indicates the level of development in an output neuron's weights. Once an output neuron is trained enough to recognize a certain class from the dataset, it will start to fire more consistently, as in **Figure 2C**, due to its high membrane potential. To quantify this consistent output neuron firing behavior, we accumulate a count of the occurrences where there are at least 8 consecutive output spikes (**Table 1**) from a specific output neuron during the 40 ms presentation period of a training sample. This count is kept for each output neuron as shown in **Figure 5**, and once an output neuron accumulates r ($r = 10$ in our case as shown in **Table 1**) such counts during training, the SNN prunes a user-defined percentage of its weights. We choose to look for 8 consecutive output spikes based on the 200 Hz output firing rate, and the 10 count threshold is a hyperparameter to control how early or late to prune an output neuron. It is worth noting that the pruning percentages are set externally in our method and they can be chosen according to the dataset, the accuracy requirement and power/latency budget of the specific applications.

We explore two different methods of pruning in this work. We use the conventional pruning method (Han et al., 2015) to prune the weights by setting their values to 0, which we also refer to pruning in this work. We also investigate a soft-pruning method (Kijsirikul and Chongkasemwongse, 2001) as an extension of conventional pruning. Instead of completely

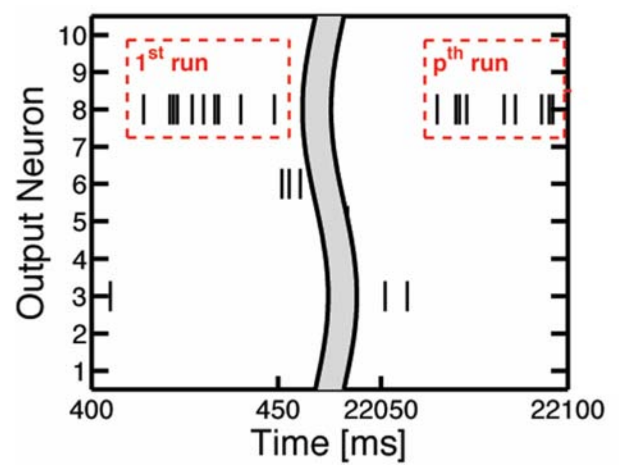


FIGURE 5 | The illustration of consecutive output spikes of 10 output neurons as a representative example. The consecutive output spikes of Neuron 8 are boxed in red.

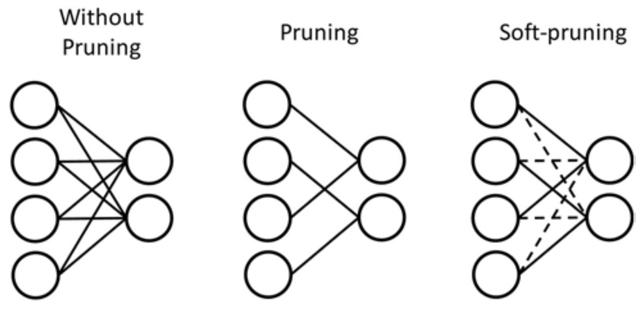


FIGURE 6 | Pruning prunes weights with values around 0 by setting these weights to 0. Much like a sparse matrix, these weights do not have to be stored in memory if their index is stored. Therefore, the synapses are physically removed to represent pruning. In contrast, soft-pruning prunes weights with values meeting certain criteria by keeping these weights constant at a certain value for the rest of training. Therefore, the pruned weights still need to be stored because they can be nonzero, but they are represented by dashed lines to indicate that they no longer need to be updated during training of the SNN.

removing the weights by setting them to 0, soft-pruning keeps the pruned weights constant at their current values for the remainder of training, or even keeping certain weights constant at the lowest or highest weight values allowed. This allows for more flexible criteria in regard to which weights are pruned, and what values they take as a result of pruning. In this work, we set the pruned weights to the lowest possible weight values, which is −1 for our network. The advantage of pruning is in reducing the representation of the weight matrix by introducing more sparsity. **Figure 6** demonstrates this by the physical removal of synapses. However, depending on the dataset, the number of weights that will be close enough to 0 to comfortably prune without losing important information can vary. While soft-pruning does not necessarily introduce more sparsity, it can allow for more weights to be pruned, thus saving computation by preventing more weight updates without drastically altering the weight distribution. **Figure 6** shows the pruned weights via soft-pruning as dashed lines to indicate that they still need to be stored in memory and participate in the testing. Soft-pruning does not increase the sparsity of weight matrix. However, since these weights are no longer updated, this can reduce energy consumption in the hardware implementation.

The usage of these two different pruning methods is dependent on the dataset to be classified. For example, the features of an image from MNIST can be separated into binary categories, i.e., the foreground and the background. In such a case, an example of soft-pruning is to prune a percentage of the lowest-valued weights of an output neuron by keeping these weight values at the lowest possible value, which for our SNN is −1. This variant of soft-pruning is analogous to learning a weight representation where the pixels representing the background take a single value, but the pixels representing the foreground can take on a range of values. Intuitively, soft-pruning results in a weight representation that does not waste resources to encode the black background pixels in MNIST in order to learn the details of the foreground, which can have varying levels of intensity due to the stroke weight of the handwriting. The top row of **Figure 7** shows an example of the learned weight visualizations of 10 representative output neurons when the SNN is trained on the MNIST dataset in three cases: without pruning, with pruning, and with soft-pruning. By the seeding of the random number generator, we control the spiking activity of all three cases so that the third output neuron (N3) is the first to meet the pruning criteria. Therefore, up to the point before N3 is pruned, the SNNs for each of the three cases have the exact same spiking activity and weight update history for all output neurons. For example, the middle row of **Figure 7** shows that N3's weight distribution is the same for all three cases. After this point, the different pruning methods between the three cases cause the weights of the output neurons between each case to develop differently.

Comparing the weight distributions for N3 in the final row of **Figure 7**, we can verify that soft-pruning is more reasonable than pruning for the MNIST dataset because it better preserves the shape of the original weight distribution, without pruning, in **Figure 7A**. In this example, we use both pruning methods to prune half of an output neuron's weights to clearly demonstrate the effect of each pruning method on the weight distribution. For pruning in **Figure 7B**, pruning 50% of the weights centered about the value 0 results in compressing a wide range of weights, shown by the space between the two dashed lines in the middle panel. Effectively, these pruned weights, most of which represent the foreground features of the MNIST dataset, are set to 0. Although the final panel of **Figure 7B** shows a somewhat binary weight distribution, which matches the binary foreground and background features of the MNIST dataset that we want to learn, the problem is that the shape of this weight distribution is drastically different than that of the weight distribution when the weights develop without pruning, as seen in the final panel of **Figure 7A**. In contrast, the effect of soft-pruning on the shape of the weight distribution, as seen in the final panel of **Figure 7C**, is minimal when compared to the case without pruning. Therefore, the pruned output neurons will produce comparable membrane

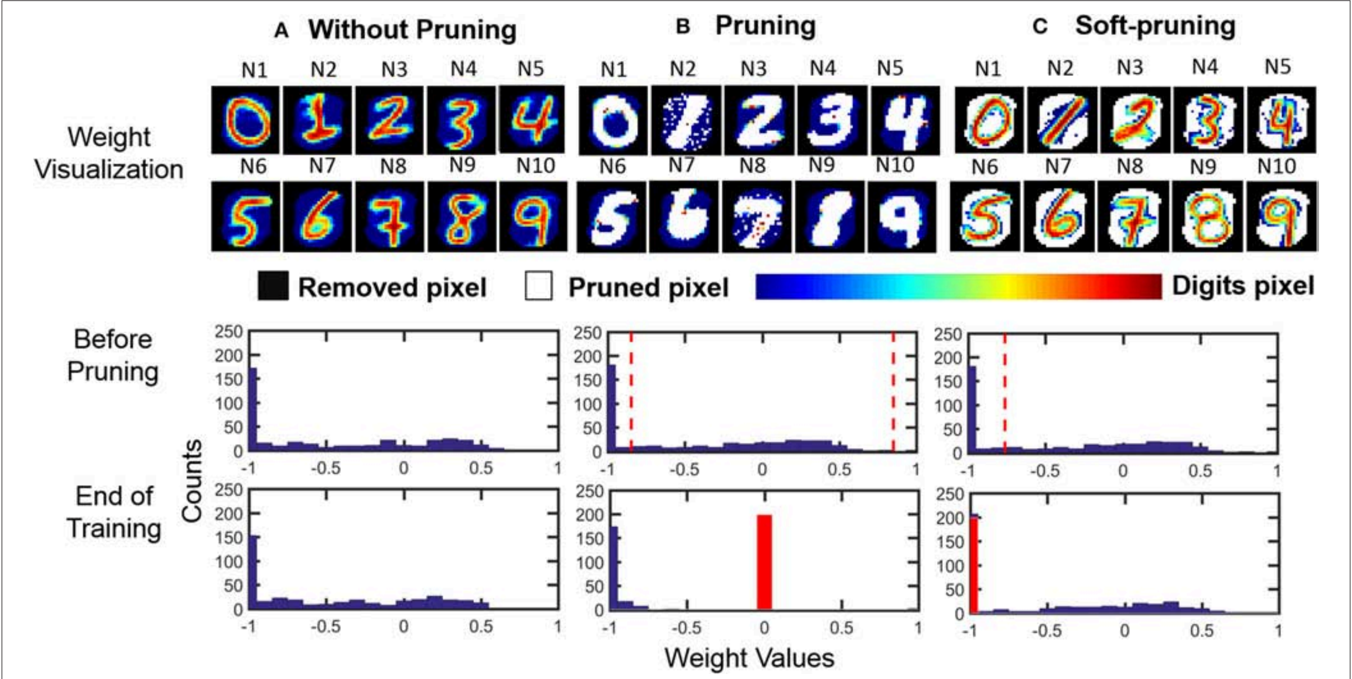


FIGURE 7 | Pruning example. (Top row) Weight visualization of the 10 representative output neurons after the SNN is trained in 3 different cases: (1) without pruning, (2) with pruning (while pruning 50% of the weights), and (3) with soft-pruning (while pruning 50% of the weights). (Middle row) Weight distribution of a representative output neuron (N3) before it is about to be pruned during training. (Bottom row) Weight distribution of the same output neuron at the end of training. It is worth to note that we only recover the removed pixels (black pixels) for visualizing the learned weights. Since those pixels are not used during training, histograms only include corresponding digit pixels (color bar) and pruned pixels (white pixels). **(A)** Without pruning, the STDP rule causes the weights for the MNIST dataset to follow a distribution where many of the weight values saturate at the lowest possible value. These low weights represent the background of MNIST training samples that are being learned by the SNN. **(B)** Pruning prunes the weights between the two dashed lines, which represent the 50% of the weights that centered about 0, and sets their values to 0 (red bar). **(C)** Soft-pruning prunes the weights to the left of the dashed line, which represent the 50% of the weights that are the lowest-valued weights, to the lowest possible value, which is at −1 (red bar). For both pruning and soft-pruning, the weights that are not pruned continue to develop for the rest of training.

potentials to the unpruned output neurons during training, resulting in balanced training between all output neurons.

With more complex datasets, e.g., color images, we might want to prune weights by setting weights around 0 to 0, or by setting weights to their current value. Han et al. demonstrate the former (Han et al., 2015). In the latter case, an interpretation can be that we set unimportant weights to their current value with the assumption that their current representation is already satisfactory for learning. Another approach is to freeze important, high-valued weights, which is a recently explored neuro-inspired concept called consolidation (Mnih et al., 2015).

RESULTS AND DISCUSSION

We simulate our SNN model, pruning and soft-pruning in MATLAB. To determine a suitable size for the training dataset, we find via **Figure 8A** that three epochs (60,000 training samples per epoch) is sufficient to reach ∼94% classification accuracy. Additionally, from **Figure 8B**, we use a 50 ms presentation period per training sample because longer presentation times show diminishing improvements in classification accuracy. **Figure 8C** shows the accuracy increases as the number of output neurons increase. However, adding output neurons will significantly increase the simulation time. Therefore, we choose to use 500 output neurons.

Following the pruning methods described in section Pruning During Training, we investigate the performance through software simulations. Simulation of classification accuracy for different p values in **Figure 9A** suggests that $r = 10$ provides the high accuracy even for very large pruning percentages (up to 80%). **Figure 9B** shows the performance of pruning and soft-pruning for varying pruning percentages when applied after training and during training. When applied after training, pruning and soft-pruning are comparable with each other until ∼50% pruning rate. After this point, the accuracy for the regular pruning method falls below ∼90% at ∼60% pruning rate, but with soft-pruning, the accuracy stays at ∼90% until ∼75% pruning rate. When each method is applied during training to save on computation of weight updates, the accuracy with pruning falls below ∼90% at around a ∼40% of pruning rate, and the accuracy with soft-pruning falls below this mark at a ∼75% of pruning rate. The performance of pruning drops much earlier than soft-pruning because pruning compresses the representation of important weights and causes uneven firing between output neurons, as mentioned in section Pruning During Training. Soft-pruning during training provides comparable accuracy to pruning after training for up to 75%

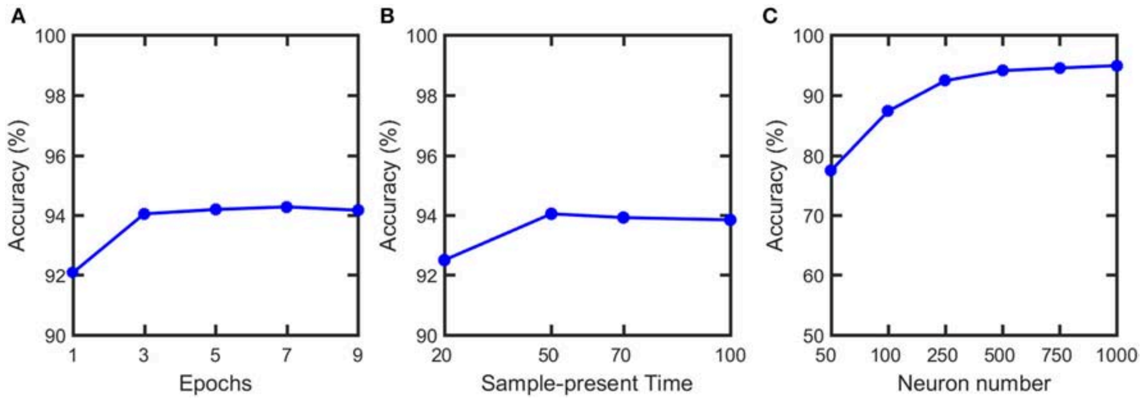


FIGURE 8 | Classification accuracy vs. **(A)** number of training epochs, **(B)** sample-present time and **(C)** output neuron numbers. Classification accuracy does not have noticeable increase after 3 epochs and 50 ms present time. Therefore, 3 epochs and 50 ms are used in the training. Although the accuracy can be further improved if neuron number increases, it will significantly increase the simulation time. Therefore, we choose to use 500 output neurons in our simulation.

pruning rate while preventing excess computation on weight updates. Additionally, when soft-pruning is applied during training, the classification accuracy is maintained at ~94% with a pruning rate up to 60%. The aim of our work is mainly energy optimization during SNN training. Therefore, soft-pruning is chosen to maintain high accuracy with larger pruning percentage, while providing significant energy reduction during training. Since soft-pruning does not completely remove synaptic weights, it is not the best way to achieve memory optimization. Alternatively, conventional pruning (Han et al., 2015) presented in this work completely removes synaptic weights and it can be used to reduce the size of memory array used for inference with a little loss in accuracy (**Figure 9B**).

We also compare the number of weight updates of conventional STDP (Song et al., 2000), STDP used in this work and STDP used in this work with 50% soft-pruning in **Table 2**. Since conventional STDP demonstrated by Song et al. bound the number of weight update of excitatory synapses ($\overline{g_a}$) between 0 and $\overline{g_{\max}}$ while our STDP bound the weights between −1 and 1, the number of weight updates of conventional STDP and our STDP are almost the same as shown in the **Table 2**. On the other hand, STDP+Soft-pruning significantly reduces the number of device updates for 50% soft pruning. In addition, soft-pruning is conceptually similar to stop learning that has been proposed in semisupervised models (Brader et al., 2007; Mostafa et al., 2016). However, there are two major differences between soft-pruning and stop-learning. Our SNN training is unsupervised. Therefore, the criterion for our soft-pruning to stop updating the synapses is when an output neuron can generate enough count of consecutive spikes to a specific class of MNIST digits (See section Pruning during training in the manuscript). Brader et al. (2007) use a semi-supervised model. Therefore, stop-learning will happen when the total current h of an output neuron is in agreement with instructor signal (target). The threshold θ is chosen to determine if the output neuron satisfies the criterion. Furthermore, our soft-pruning stops updating part of the synapses of an output neuron depending on the pruning percentage the user set. This means that the un-pruned synapses still can be updated for the rest of the training. However, Brader et al. stop updating all the synapses of an output neuron once the stop-learning criterion is satisfied.

TABLE 2 | The number of weight updates of conventional STDP (Song et al., 2000), STDP used in this work with and without 50% soft-pruning.

	# of weight updates
Conventional STDP (Song et al., 2000)	649289638
STDP (this work)	648669156
STDP (this work) + 50% Soft-pruning	357656929

Our classification accuracy is comparable to previous software implementations of unsupervised learning for the MNIST dataset with SNNs (**Table 3**). As can be seen from the table, multilayer SNNs (Diehl and Cook, 2015a; Kheradpisheh et al., 2017; Tavanaei and Maida, 2017; Ferré et al., 2018) generally have higher accuracy than single layer SNNs. However, the works with accuracy higher than 95% (Kheradpisheh et al., 2017; Tavanaei and Maida, 2017; Ferré et al., 2018) all require using multiple convolution and pooling layers, and other complex processing techniques, which are difficult to implement in hardware. Compared to the SNNs without convolution layers, our classification accuracy is much higher than previous single layer SNNs (Nessler et al., 2013; Al-Shedivat et al., 2015) and achieves performance very close to Diehl and Cook (2015a) with much fewer neurons and synapses. Our single layer SNN architecture does not require complex processing and is particularly suitable for easy hardware implementation. Differing from all previous approaches, we present a novel pruning method to reduce the number of updates to network parameters during SNN training. Hence, despite only part of the synapses in our network needing to be updated during training, our SNN still maintains a high classification accuracy with up a 75% pruning rate. Therefore, our pruning scheme can potentially reduce the energy consumption and

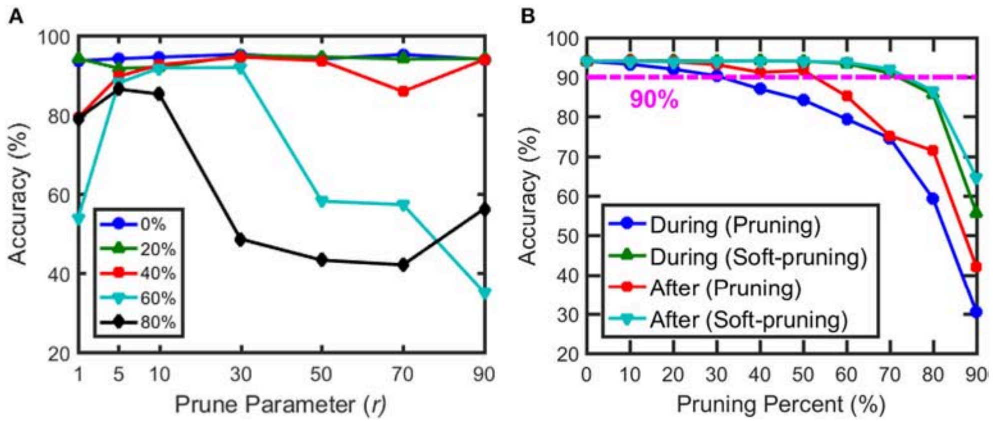


FIGURE 9 | (A) Classification accuracy vs. prune parameter (*r*) for varying pruning percentages. Prune parameter is the criterion to decide when to prune for each neuron during training. **(B)** Classification accuracy vs. pruning percentage for pruning and soft-pruning when applied during training and after training. The data points are taken in steps of 10%. The dashed line represents classification accuracy of 90%. Soft-pruning during the training performs better than pruning especially for high pruning percentages. Soft-pruning maintains > 90% up to 75% pruning percentage while pruning falls below 90% at only 40% pruning. Although we focus on pruning during training, we also present results from pruning weights after training as a baseline for previously established pruning methods from the literature. The parameters used in the simulation are specified in **Table 1**.

TABLE 3 | Classification accuracy comparison between this work and the state-of-the-art software demonstrations of unsupervised learning of SNNs on the MNIST dataset.

Architecture	Complex Processing	Learning rule	#Neurons/synapses	Pruning during training	Performance
Spiking deep neural network (Kheradpisheh et al., 2017)	Convolution, DOG filter, Pooling	Simplified STDP	N/A	None	98.4%
Multi layer (Ferré et al., 2018)	Convolution, Pooling, Dropout	Binary STDP	N/A	None	98.49%
Three layer (Tavanaei and Maida, 2017)	Convolution, Pooling	Probabilistic STDP	N/A	None	98.36%
Two layer (Diehl and Cook, 2015a)	None	Exponential STDP	7,184/5,017,600	None	95%
One layer (Al-Shedivat et al., 2015)	Population Coding	Probabilistic STDP	1,696/200,704	None	78.4%
One layer (Nessler et al., 2013)	Population Coding	Exponential STDP	808/70,800	None	80.14%
One layer (this work)	**None**	**Simplified STDP**	**898/∼199,000**	**Yes (∼75%)**	**94.05%**

The table lists the complex processing techniques used, the learning rule, and the #Neurons/synapses used in each work. The table also indicates if pruning during training is involved in the work. The numbers of neurons are counted by summing the input and output neurons.

training time in hardware implementation. The simple one-layer SNN architecture and STDP rule proposed in our work mainly focus on demonstrating the idea of pruning during the training. Scaling our SNN algorithm to larger datasets can be achieved by modifying the network architecture in several approaches such as by adding more fully connected layers (Diehl et al., 2015b; Lee et al., 2016; O'connor and Welling, 2016) or convolutional layers (Diehl et al., 2015b; Lee et al., 2016; Tavanaei and Maida, 2017; Kulkarni and Rajendran, 2018), adjusting learning rule and involving the supervision (Kulkarni and Rajendran, 2018).

Our single layer SNN network (**Figure 10A**) can be directly mapped to a crossbar array based on eNVM devices (**Figure 10B**) to perform online learning. The input of the network is decoded into a Poisson spike train based on the pixel intensity (see section Input layer for details) and it can be mapped to the input voltage spikes of the crossbar array (**Figure 10A**). There are many demonstrations showing that eNVM devices can have multilevel conductance states to emulate analog weight tuning (Jo et al., 2010; Kuzum et al., 2011). Therefore, the weights in the SNN can be represented using the conductance of eNVM devices. Since the weights in our network is ranging from −1 to 1, there are two ways to use device conductance to represent the weights. One approach could be using a single device to represent a synaptic weight. The weights in the network are linearly transformed to the conductance range as shown in Equation (6) for the hardware implementation (Serb et al., 2016; Kim et al., 2018; Li et al., 2018; Oh et al., 2018; Shi et al., 2018).

$$G = W\frac{(G_{\max} - G_{\min})}{2} + \frac{(G_{\max} + G_{\min})}{2} \qquad (6)$$

An alternative approach could be using of two devices as one synaptic weight as shown in previous literature (Burr et al., 2015; Li et al., 2018). Both positive and negative

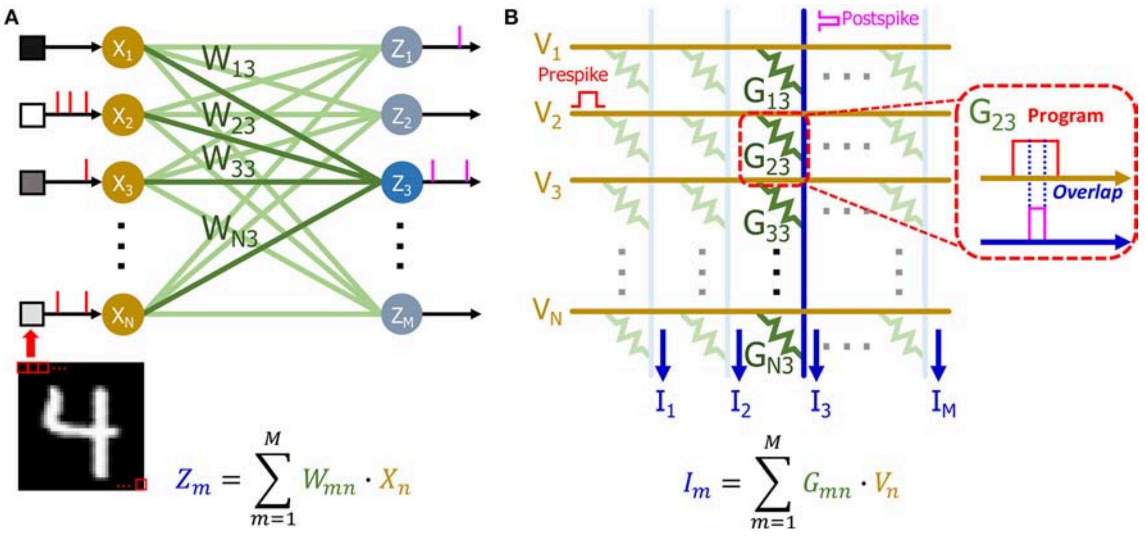

A
B
Postspike
Prespike
Program
Overlap
$Z_m = \sum_{m=1}^{M} W_{mn} \cdot X_n$
$I_m = \sum_{m=1}^{M} G_{mn} \cdot V_n$

FIGURE 10 | (A) Schematic of SNN with *n* input neurons and *m* output neurons. The pixel intensities of input image are decoded into passion spiking training and fed to the input of the network. The weights (W_{13}, W_{23}, …, W_{N3}) of output neuron Z_3 has been highlighted. **(B)** Schematic of a crossbar array based on eNVM devices. The input of **(A)** can be mapped to the voltage. The weights (W_{13}, W_{23}, …, W_{N3}) are mapped to the conductance (G_{13}, G_{23}, …, G_{N3}) of the devices. The weighted sum can be obtained by measuring the current at the end of each column. The postspike pulses are generated based on the weighted sums (*I*). The overlap of pre and post spike pulses as shown in callout window programs the device to different conductance states.

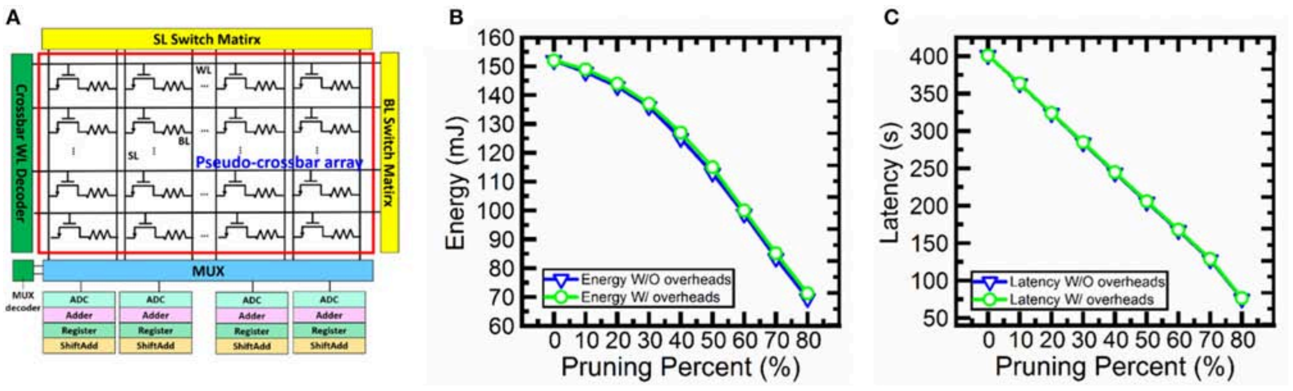


FIGURE 11 | (A) Analog synaptic core uses a single cell with multi-level conductance states to represent one synaptic weight. One transistor is added to each cell in order to avoid sneak path problem. The crossbar wordline (WL) decoder can activate all WLs, bitline (BL) read out the weighted sum results, and source line (SL) can be used to perform weight update. Multiplexer (MUX) is used to share the neuron circuitry. The neuron circuit contains analog-to-digital converters (ADCs), adders, registers and shift adders, which are used to perform weighted sum. **(B)** The energy and **(C)** latency without and with overhead estimation for soft-pruning from 10 to 80% with a step of 10% using SNN+NeuroSim. Without overheads (W/O overheads) results mean that flagging mechanism is implemented in software. With overheads (W/ overheads) results mean that flagging mechanism is implemented in hardware.

weights can be represented by taking the difference between conductance of two devices ($G = G^{+} - G^{-}$). The weighted sum operation for calculating membrane potential (see section Output layer for details) can be calculated in a single step by accumulating the current flowing through each column in the crossbar array (Eryilmaz et al., 2016). Our STDP weight update rule can be realized by overlapping of the prespike and postspike pulses (**Figure 10B**) to program the device to different conductance levels, as shown in previous demonstrations (Kuzum et al., 2011, 2012).

In order to implement pruning in hardware, the pruned cells need to be flagged to prevent them from being updated further. One solution is to use an extra binary device associated with each eNVM synaptic weight to serve as a hardware pruning flag. This binary device is initially programmed to "0" (the lowest conductance state), to indicate that the cell has not been pruned. We update the pruning flag of an output neuron's weights to "1" (the highest conductance state) when it has been pruned during training. Before the weight update, we read the hardware flag of the winning neuron's weight to decide whether or not to update. The weights are only pruned once during the entire training. As a result, each hardware flag is just written once and hence the energy overhead will be negligible. However, the hardware pruning flag will slightly increase the area of the array. If the size of the array is crucial for a system, an alternative way can be used to implement the hardware flag without area overhead. The pruned cells can be reset to a very low conductance state with additional reset current (Arita et al., 2015; Xia et al., 2017). Such cells generally require reforming to be programmed to a multi-level conductance state regime again (Wong et al., 2012). Therefore, the pruned cells will not be further updated during training and we can use its very low conductance state as pruning flag.

In order to confirm the feasibility of the proposed hardware implementation of pruning during SNN training. We perform circuit-level benchmarking simulations with NeuroSim (Chen et al., 2018) to evaluate the performance of a full system of analog synaptic core as shown in **Figure 11A**. NeuroSim is a C++ based simulator with hierarchical organization starting from experimental device data and extending to array architectures with peripheral circuit modules and algorithm-level neural network models (Chen et al., 2018). We develop a SNN platform for NeuroSim (SNN+NeuroSim). SNN+NeuroSim can simulate circuit-level performance metrics (area, energy and latency) at run-time of online learning using eNVM arrays. We implement the hardware flagging mechanism of pruning in SNN+NeuroSim and estimate energy and latency overheads caused by flagging mechanism. **Figures 11B,C** show energy and latency without and with overheads due to pruning. The results show that the energy and latency can be significantly decreased as the pruning percentages increase. The results also suggest that energy consumption and latency do not significantly increase due to the overheads associated with the hardware flag for the pruning percentages from 10 to 80%.

CONCLUSION

In this work, we first demonstrate a low-complexity single layer SNN training model for unsupervised learning on MNIST. We then develop a new method to prune during training for SNNs. Our pruning scheme exploits the output spike firing of the SNN to reduce the number of weight updates during network training. With this method, we investigate the impact of pruning and soft-pruning on classification accuracy. We show that our SNN can maintain high classification accuracy (~90%) on the MNIST dataset and the network can be extensively pruned (75% pruning rate) during training. We also discuss and simulate the possible hardware implementation of our SNN and pruning algorithm with eNVM crossbar arrays using SNN+NeuroSim. Our algorithmic optimization approach can be applied to improve network level energy efficiency of other SNNs with eNVM arrays for in-memory computing applications, enabling online learning of SNNs in power-limited settings.

AUTHOR CONTRIBUTIONS

YS, LN, and DK conceived the idea. YS and LN developed the pruning algorithm. YS, LN, and SO implemented unsupervised learning neural network simulation, and analysis the data obtained from the simulation. All authors wrote the manuscript, discussed the results and commented on the manuscript. DK supervised the work.

ACKNOWLEDGMENTS

The authors acknowledge support from the Office of Naval Research Young Investigator Award (N00014161253), National Science Foundation (ECCS-1752241, ECCS-1734940) and Qualcomm FMA Fellowship for funding this research.

REFERENCES

Alibart, F., Zamanidoost, E., and Strukov, D. B. (2013). Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nat. Commun.* 4:2072. doi: 10.1038/ncomms3072

Al-Shedivat, M., Naous, R., Cauwenberghs, G., and Salama, K. N. (2015). Memristors empower spiking neurons with stochasticity. *IEEE J. Emerg. Select. Topics Circuits Syst.* 5, 242–253. doi: 10.1109/JETCAS.2015.2435512

Ankit, A., Sengupta, A., Panda, P., and Roy, K. (2017). "Resparc: a reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017: ACM* (Austin, TX), 27.

Arita, M., Takahashi, A., Ohno, Y., Nakane, A., Tsurumaki-Fukuchi, A., and Takahashi, Y. (2015). Switching operation and degradation of resistive random access memory composed of tungsten oxide and copper investigated using in-situ TEM. *Sci. Rep.* 5:17103. doi: 10.1038/srep17103

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput.* 19, 2881–2912. doi: 10.1162/neco.2007.19.11.2881

Burr, G. W., Shelby, R. M., Sidler, S., Di Nolfo, C., Jang, J., Boybat, I., et al. (2015). Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE Trans. Electron Devices* 62, 3498–3507. doi: 10.1109/TED.2015.2439635

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Chen, P.-Y., Peng, X., and Yu, S. (2018). NeuroSim: a circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput. Aided Design Integrat. Circuits Syst.* 37, 3067–3080. doi: 10.1109/TCAD.2018.2789723

Choi, S., Sheridan, P., and Lu, W. D. (2015). Data clustering using memristor networks. *Sci. Rep.* 5:10492. doi: 10.1038/srep10492

Cruz-Albrecht, J. M., Yung, M. W., and Srinivasa, N. (2012). Energy-efficient neuron, synapse and STDP integrated circuits. *IEEE Trans. Biomed. Circuits Syst.* 6, 246–256. doi: 10.1109/TBCAS.2011.2174152

Deger, M., Helias, M., Rotter, S., and Diesmann, M. (2012). Spike-timing dependence of structural plasticity explains cooperative synapse formation in the neocortex. *PLoS Comput. Biol.* 8:e1002689. doi: 10.1371/journal.pcbi.1002689

Deger, M., Seeholzer, A., and Gerstner, W. (2017). Multicontact co-operativity in spike-timing-dependent structural plasticity stabilizes networks. *Cerebral. Cortex* 28, 1396–1415. doi: 10.1093/cercor/bhx339

Diehl, P. U., and Cook, M. (2015a). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook,M., Liu, S.-C., and Pfeiffer, M. (2015b). "Fastclassifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN): IEEE* (Killarney), 1–8.

Eryilmaz, S. B., Neftci, E., Joshi, S., Kim, S., Brightsky, M., Lung, H.-L., et al. (2016). Training a probabilistic graphical model with resistive switching electronic synapses. *IEEE Trans. Electron Devices* 63, 5004–5011. doi: 10.1109/TED.2016.2616483

Ferré, P., Mamalet, F., and Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based STDP. *Front. Comput. Neurosci.* 12:24. doi: 10.3389/fncom.2018.00024

Ge, R., Wu, X., Kim, M., Shi, J., Sonde, S., Tao, L., et al. (2017). Atomristor: nonvolatile resistance switching in atomic sheets of transition metal dichalcogenides. *Nano Lett.* 18, 434–441. doi: 10.1021/acs.nanolett.7b04342

Gupta, A., and Long, L. N. (2007). "Character recognition using spiking neural networks," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on: IEEE* (Orlando, FL), 53–58.

Han, S., Pool, J., Tran, J., and Dally, W. (2015). "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems* (Montreal, QC), 1135–1143.

Iglesias, J., and Villa, A. E. (2007). Effect of stimulus-driven pruning on the detection of spatiotemporal patterns of activity in large neural networks. *BioSystems* 89, 287–293. doi: 10.1016/j.biosystems.2006.05.020

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* 10, 1297–1301. doi: 10.1021/nl904092h

Kappel, D., Habenschuss, S., Legenstein, R., and Maass, W. (2015). "Synaptic sampling: a bayesian approach to neural network plasticity and rewiring," in *Advances in Neural Information Processing Systems* (Montreal, QC), 370–378.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., Masquelier, T. (2017). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw*. (2017). doi: 10.1016/j.neunet.2017.12.005

Kijsirikul, B., and Chongkasemwongse, K. (2001). "Decision tree pruning using backpropagation neural networks," in *Proceedings of IEEE International Conference on Neural Networks* (Washington, DC), 1876–1880.

Kim, H., Kim, T., Kim, J., and Kim, J.-J. (2018). Deep neural network optimized to resistive memory with nonlinear current-voltage characteristics. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* 14:15. doi: 10.1145/3145478

Kulkarni, S. R., and Rajendran, B. (2018). Spiking neural networks for handwritten digit recognition-Supervised learning and network optimization. *Neural Networks* 103, 118–127. doi: 10.1016/j.neunet.2018.03.019

Kuzum, D., Jeyasingh, R. G., Lee, B., and Wong, H.-S. P. (2011). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* 12, 2179–2186. doi: 10.1021/nl201040y

Kuzum, D., Jeyasingh, R. G. D., Yu, S., and Wong, H. S. P. (2012). Low-energy robust neuromorphic computation using synaptic devices. *IEEE Trans. Electron Devices* 59, 3489–3494. doi: 10.1109/TED.2012.2217146

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Li, C., Hu, M., Li, Y., Jiang, H., Ge, N., Montgomery, E., et al. (2018). Analogue signal and image processing with large memristor crossbars. *Nat. Electron.* 1, 52–59. doi: 10.1038/s41928-017-0002-z

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. doi: 10.1038/nature14236

Mostafa, H., Mayr, C., and Indiveri, G. (2016). "Beyond spike-timing dependent plasticity in memristor crossbar arrays," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS): IEEE*, 926–929.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., and Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* 7:272. doi: 10.3389/fnins.2013.00272

Nessler, B., Pfeiffer, M., Buesing, L., and Maass, W. (2013). Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Comput. Biol.* 9:e1003037. doi: 10.1371/journal.pcbi.1003037

O'connor, P., and Welling, M. (2016). Deep spiking networks. *arXiv preprint arXiv*:1602.08323.

Oh, S., Shi, Y., Liu, X., Song, J., and Kuzum, D. (2018). Drift-enhanced unsupervised learning of handwritten digits in spiking neural network with PCM synapses. *IEEE Electron Device Lett.* 39, 1768–1771. doi: 10.1109/LED.2018.2872434

Panda, P., Srinivasan, G., and Roy, K. (2017a). Convolutional spike timing dependent plasticity based feature learning in spiking neural networks. *arXiv preprint arXiv:*1703.03854.

Panda, P., Srinivasan, G., and Roy, K. (2017b). "EnsembleSNN: distributed assistive STDP learning for energy-efficient recognition in spiking neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*: *IEEE*, 2629–2635.

Perfors, A., Tenenbaum, J. B., Griffiths, T. L., and Xu, F. (2011). A tutorial introduction to bayesian models of cognitive development. *Cognition* 120, 302–321. doi: 10.1016/j.cognition.2010.11.015

Prezioso, M., Merrikh-Bayat, F., Hoskins, B., Adam, G., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521:61–64. doi: 10.1038/nature14441

Rathi, N., Panda, P., and Roy, K. (2018). STDP based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition. *IEEE Trans. Comput. Aided Design Integrat. Circuits Syst.* 38, 668–677. doi: 10.1109/TCAD.2018.2819366

Sengupta, A., Parsa, M., Han, B., and Roy, K. (2016). Probabilistic deep spiking neural systems enabled by magnetic tunnel junction. *IEEE Trans. Electr. Devices* 63, 2963–2970. doi: 10.1109/TED.2016.2568762

Serb, A., Bill, J., Khiat, A., Berdan, R., Legenstein, R., and Prodromakis, T. (2016). Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses. *Nat. Commun.* 7:12611. doi: 10.1038/ncomms12611

Shi, Y., Nguyen, L., Oh, S., Liu, X., Koushan, F., Jameson, J. R., et al. (2018). Neuroinspired unsupervised learning and pruning with subquantum CBRAM arrays. *Nat. Commun.* 9:5312. doi: 10.1038/s41467-018-07682-0

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Spiess, R., George, R., Cook, M., and Diehl, P. U. (2016). Structural plasticity denoises responses and improves learning speed. *Front. Comput. Neurosci.* 10:93. doi: 10.3389/fncom.2016.00093

Srinivasan, G., Sengupta, A., and Roy, K. (2016). Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip STDP learning. *Sci. Rep.* 6:29545. doi: 10.1038/srep29545

Tavanaei, A., and Maida, A. S. (2017). "Multi-layer unsupervised learning in a spiking convolutional neural network," in *Neural Networks (IJCNN), International Joint Conference on: IEEE* (Anchorage, AK), 2023–2030.

Tavanaei, A.,Masquelier, T., and Maida, A. S. (2016). "Acquisition of visual features through probabilistic spike-timing-dependent plasticity," in *Neural Networks (IJCNN), 2016 International Joint Conference on: IEEE* (Vancouver, BC), 307–314.

Vincent, A. F., Larroque, J., Zhao, W., Romdhane, N. B., Bichler, O., Gamrat, C., et al. (2014). "Spin-transfer torque magnetic memory as a stochastic memristive synapse," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS): IEEE* (Melbourne, VIC), 1074–1077.

Wong, H.-S. P. (2018). "The end of the road for 2 Dscaling of silicon CMOS and the future of device technology," in *2018 76th Device Research Conference (DRC): IEEE* (Santa Barbara, CA), 1–2.

Wong, H.-S. P., Lee, H.-Y., Yu, S., Chen, Y.-S., Wu, Y., Chen, P.-S., et al. (2012). Metal-oxide RRAM. *Proc. IEEE.* 100, 1951–1970. doi: 10.1109/JPROC.2012.2190369

Xia, L., Liu, M., Ning, X., Chakrabarty, K., and Wang, Y. (2017). "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *Proceedings of the 54th Annual Design Automation Conference 2017*: *ACM*, 33.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Neuromorphic Hardware Learns to Learn

Thomas Bohnstingl[1*†‡], *Franz Scherr*[1‡], *Christian Pehle*[2], *Karlheinz Meier*[2] and *Wolfgang Maass*[1]

[1] Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria, [2] Kirchhoff-Institute for Physics, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany

Hyperparameters and learning algorithms for neuromorphic hardware are usually chosen by hand to suit a particular task. In contrast, networks of neurons in the brain were optimized through extensive evolutionary and developmental processes to work well on a range of computing and learning tasks. Occasionally this process has been emulated through genetic algorithms, but these require themselves hand-design of their details and tend to provide a limited range of improvements. We employ instead other powerful gradient-free optimization tools, such as cross-entropy methods and evolutionary strategies, in order to port the function of biological optimization processes to neuromorphic hardware. As an example, we show these optimization algorithms enable neuromorphic agents to learn very efficiently from rewards. In particular, meta-plasticity, i.e., the optimization of the learning rule which they use, substantially enhances reward-based learning capability of the hardware. In addition, we demonstrate for the first time Learning-to-Learn benefits from such hardware, in particular, the capability to extract abstract knowledge from prior learning experiences that speeds up the learning of new but related tasks. Learning-to-Learn is especially suited for accelerated neuromorphic hardware, since it makes it feasible to carry out the required very large number of network computations.

Keywords: spiking neural networks, learning-to-learn, markov decision processes, multi-armed bandits, neuromorphic hardware, HICANN-DLS, meta-plasticity, transfer learning

OPEN ACCESS

Edited by:
Yansong Chua,
Institute for Infocomm Research (A∗STAR), Singapore

Reviewed by:
Sadique Sheik,
AiCTX AG, Switzerland
Garibaldi Pineda García,
University of Sussex, United Kingdom

****Correspondence:***
Thomas Bohnstingl
boh@zurich.ibm.com

†Present Address:
Thomas Bohnstingl,
IBM Research - Zurich, Rüschlikon, Switzerland

‡These authors have contributed equally to this work

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *31 January 2019*
Accepted: *29 April 2019*
Published: *21 May 2019*

Citation:
Bohnstingl T, Scherr F, Pehle C, Meier K and Maass W (2019) Neuromorphic Hardware Learns to Learn. Front. Neurosci. 13:483. doi: 10.3389/fnins.2019.00483

1. INTRODUCTION

The computational substrate that the human brain employs to carry out its computational functions, is given by networks of spiking neurons (SNNs). There appear to be numerous reasons for evolution to branch off toward such a design. For example, networks of such neurons facilitate a distributed scheme of computation, intertwined with memory entities, thereby overcoming known disadvantages in contemporary computer designs such as the von Neumann bottleneck. Importantly, the human brain serves as an inspiration for a power efficient learning machine, solving demanding computational tasks while consuming little resources. A characteristic property that makes energy efficient computation possible is the distinct communication among these neurons. In particular, neurons do not need to produce an output at all times. Instead, information is integrated over time and communicated sparsely using a format of discrete events, "spikes."

The connectivity structure, the development of computational functions in specific brain regions, as well as the active learning algorithms are all subject to an evolutionary process. In particular, evolution has shaped the human brain and successfully formed a learning machine, capable of carrying out a range of complex computations. In close connection to this, a characteristic property of learning processes in humans is the ability to take advantage of previous, related experiences and use them in novel tasks. Indeed, humans show both, the ability to quickly adapt to new challenges in various domains, and the ability to transfer prior acquired knowledge about different, but related tasks to new, potentially unseen ones (Taylor and Stone, 2009; Robert Canini et al., 2010; Wang and Zheng, 2015).

One strategy to investigate the benefit of a knowledge transfer between different, but related learning tasks is to impose a so-called Learning-to-Learn (L2L) optimization. L2L employs task-specific learning algorithms, but also tries to mimic the slow evolutionary and developmental processes that have prepared brains for the learning tasks humans have to face. In particular, L2L introduces a nested optimization procedure, consisting of an inner loop and an outer loop. In the inner loop, specific tasks are learned, while an additional outer loop aims to optimize the learning performance on a range of different tasks. This concept gave rise to an interesting body of work (Hochreiter et al., 2001; Wang et al., 2016; Finn et al., 2017) and showed that one can endow artificial learning systems with transfer learning capabilities. Recently, this concept was also extended to networks of spiking neurons. In a study by Bellec et al. (2018) it is shown that a biologically inspired circuit can encode prior assumptions about the tasks it will encounter.

Usually, one takes advantage of the availability of gradient information to facilitate optimization, here instead, we employ powerful gradient-free optimization algorithms in the outer loop that emulate the evolutionary process. In particular, we demonstrate the benefits of evolutionary strategies (ES) (Rechenberg, 1973) and cross entropy methods (CE) (Rubinstein, 1997), as they are able to deal with noisy function evaluations and perform in high-dimensional spaces. In the inner loop, on the other hand, we consider reinforcement learning problems (RL problems), such as Markov Decision Processes and Multi-armed bandits. Problems of this type appear quite often in general and therefore, a rich literature has emerged. However, it still remains that learning from rewards is particularly inefficient, as the feedback is given by a single scalar quantity, the reward. We show that by employing the concept of L2L we can produce agents that learn efficiently from rewards and exploit previous experiences on related, new tasks.

As another novelty, we implement the learning agent on a neuromorphic hardware (NM hardware). Specialized hardware of this type has emerged by taking inspiration of principles of brain computation, with the intent to port the advantages of distributed and power efficient computation to silicon chips (Mead, 1990). This holds the great promise to install artificial intelligence in devices without cloud connection and/or limited resource. Numerous architectures have been proposed that are either based on analog, digital or mixed-signal approaches: (Schemmel et al., 2010; Furber et al., 2014; Furber, 2016; Pantazi et al., 2016; Aamir et al., 2018; Ambrogio et al., 2018; Davies et al., 2018; Wunderlich et al., 2018). We refer to Schuman et al. (2017) for a survey on neuromorphic systems.

In order to further enhance the learning capabilities of NM hardware, we exploit the adjustability of the employed neuromorphic chip and consider the use of meta-plasticity. In other words, we evolve a highly configurable plasticity rule that is responsible for learning in the network of spiking neurons. To this end, we represent the plasticity rule as a multilayer perceptron (section 2.5.2) and demonstrate that this approach can significantly boost learning performance as compared to the level that is achieved by plasticity rules that we derive from general algorithms, see section 3.3.

NM hardware is especially well-suited for L2L because it renders the large number of simulations that need to be carried out feasible. Spiking neurons that are simulated on NM hardware typically exhibit accelerated dynamics as compared to their biological counterparts. In addition, the chosen neuromorphic hardware allows to emulate both, the RL environment as well as the learning algorithm at the same acceleration factor and hence, one unlocks the full potential of the specialized neuromorphic chip.

First, in section 2 we will discuss our approaches and methods, as well as the set of tools (https://github.com/bohnstingl/Neuromorphic_Hardware_learns_to_learn) that was used in our experiments. In particular, the employed NM hardware is discussed in section 2.3. Then, in section 3.1 we will exhibit the increase in performance and learning speed that we obtained on NM hardware for the conducted tasks and discuss which gradient-free algorithms worked best for our setting. Afterwards, we discuss in section 3.3 that performance can be further increased by the adoption of a highly customizable learning rule, i.e., meta-plasticity, that is shaped through L2L, and discuss its relevance in transfer learning. We also discuss the impact in terms of simulation time thanks to the underlying NM hardware. Finally, we conclude our findings and results in section 4.

2. METHODS AND MATERIALS

This section provides the technical details to the conducted experiments. First, we describe the background for L2L in section 2.1, and discuss the gradient-free optimization techniques that are employed. Subsequently, we provide details to the reinforcement learning tasks that we considered (section 2.2).

Since the agent that interacts with the RL environments is implemented on a NM hardware, we discuss the corresponding chip in section 2.3. We exhibit the network structure that we used throughout all our experiments in section 2.4. Subsequently, we provide details to the learning algorithms that we used in section 2.5 and discuss methods for analysis.

2.1. Learning-to-Learn and Gradient-free Optimization

The goal of Learning-to-Learn is to enhance a learning systems' capability to learn. In models of neural networks, learning performance can be enhanced by several methods. For example, one can optimize hyperparameters that affect the learning procedure or optimize the learning procedure as such. Often, this optimization is carried out manually and involves a lot of domain knowledge. Here instead, we evolve suitable hyperparameters as well as learning algorithms automatically by the means of L2L.

In particular, L2L introduces a nested optimization that consists of two loops: an inner loop and an outer loop as displayed in **Figure 1**. In the inner loop, one considers a particular task C_i in which the model $\mathcal{N}$ has to use its learning capabilities to succeed. The outer loop, on the other hand, is responsible to adapt the learning procedure that is used by $\mathcal{N}$ such that it becomes better at learning tasks in a given family $\mathcal{F}$ that share some similar concepts. To express the quality of the learning procedure, we introduce a learning fitness $f(C_i; \Theta)$ that measures how well the model $\mathcal{N}$ can learn a task C_i, e.g., what is the cumulative reward that was achieved. This learning fitness depends on both the specific task that is being learnt, as well as the hyperparameters Θ that characterize the learning procedure. We write the goal of L2L is then as an optimization problem, where we want to find hyperparameters that yield the best learning procedure for tasks in the family $\mathcal{F}$:

$$\max_{\Theta'} \mathbb{E}_{C\sim\mathcal{F}}\left[f(C; \Theta')\right]. \tag{1}$$

In practice, the family of tasks could be comprised of infinite tasks and hence, the expectation in Equation (1) is approximated using batches of N different tasks: $\mathbb{E}_{C\sim\mathcal{F}}\left[f(C; \Theta)\right] \approx \frac{1}{N}\sum_{i=1}^{N} f(C_i; \Theta) = \widehat{f}(\Theta)$. As a result of considering different tasks C_i in the inner loop each time, the hyperparameters can only assume task independent concepts that are shared throughout the family. In fact, one can consider L2L as an optimization that happens on two different timescales: fast learning of single tasks in the inner loop, and a slower learning process that adapts hyperparameters in order to boost learning on the entire family of learning tasks.

The L2L scheme allows separating the learning process in the inner loop from the optimization algorithms that work in the outer loop. We used Q-Learning and Meta-Plasticity to implement learning in the inner loop (discussed in section 2.5),

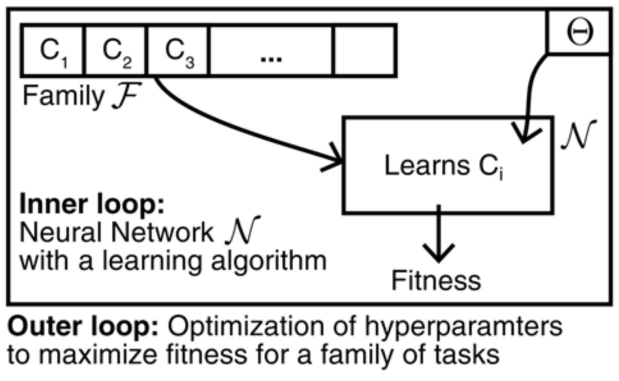

C1
C2
C3
...
Θ
Family $\mathcal{F}$
Learns Ci
$\mathcal{N}$
Inner loop:
Neural Network $\mathcal{N}$
with a learning algorithm
Fitness
Outer loop: Optimization of hyperparamters
to maximize fitness for a family of tasks

FIGURE 1 | Learning-to-Learn scheme. Learning-to-Learn introduces a nested optimization with two loops. In the inner loop a model is required to perform a learning task C_i from a family $\mathcal{F}$. The learning capabilities of the model are influenced by hyperparameters Θ that are optimized in the outer loop in order to maximize the average learning fitness over the entire task family $\mathcal{F}$.

while at the same time, we considered several gradient-free optimization techniques in the outer loop. The requirements for a well-suited optimization algorithm in the outer loop are the ability to operate in a high-dimensional parameter space, the ability to deal with noisy fitness evaluations, the ability to find a good final solution and the ability to do so using a small number of fitness evaluations. Due to this broad set of requirements, the choice of the outer loop algorithm is non-trivial and needs to be adjusted based on the task family that is considered in the inner loop. We selected a set of gradient-free optimization techniques such as cross-entropy methods, evolutionary strategies, numerical gradient-descent as well as a parallelized variation of simulated annealing. In the following, we provide a brief outline of the algorithms used and refer to the corresponding literature. For the concrete implementation, we employ a L2L software framework that provides several such optimization methods (Subramoney et al., 2019). In particular, the L2L optimization is carried out on a Linux-based host computer, whereas the inner loop is simulated in its entirety on the later discussed neuromorphic hardware, section 2.3.

2.1.1. Cross-entropy (CE) (Rubinstein, 1997)

In each iteration, this algorithm fits a parameterized distribution $p(\cdot; \phi)$ to the set of n best-performing hyperparameters in terms of maximum likelihood. In the subsequent step, new hyperparameters are sampled from this distribution and evaluated. Afterwards, the procedure starts over again until a stopping criterion is met. Through this process, the algorithm tries to find a region of individuals where the performance is high on average. We used a univariate Gaussian distribution with a dense covariance matrix.

2.1.2. Evolution Strategies (ES) (Rechenberg, 1973)

In each iteration, this algorithm maintains base hyperparameters Θ which are perturbed by random deviations ϵ to form a new set of n hyperparameters. This set is then evaluated and ranked by their fitness. In a subsequent step, the perturbations are weighted according to their rank to produce a direction of increasing fitness, which is used to update the base hyperparameters. Similar to Cross-entropy, ES also finds a region of hyperparameters with high fitness, rather than just a single one. Note that many variations of this algorithm have been proposed that differ for example in the way how the ranking or how the perturbations are computed (Salimans et al., 2017). In particular, we used Algorithm 1 from Salimans et al. (2017).

2.1.3. Simulated Annealing (SA) (Kirkpatrick et al., 1983)

In each iteration, the algorithm maintains hyperparameters Θ and a temperature T. The hyperparameters are perturbated with a random ϵ, whose size depends on the temperature T, and are evaluated later. The fitness of the unperturbed hyperparameters Θ is then compared with the perturbated hyperparameters Θ'. The Θ' replaces Θ with a probability of $\min\left(1, \exp\left(-\left(\widehat{f}(\Theta') - \widehat{f}(\Theta)\right)/T\right)\right)$. In the next step, the temperature is decreased following a predefined schedule and the new hyperparameters get perturbed. In contrast to the other methods discussed before, a single set of hyperparameters is the result. In our experiments, we simultaneously perform a number of parallel SA optimizations, using a linear temperature decay.

2.1.4. Numerical Gradient-Descent (GD)

In each iteration, the algorithm maintains hyperparameters Θ which are perturbed randomly in many directions and then evaluated. Subsequently, the gradient is numerically estimated and an ascending step on the fitness landscape is performed.

2.2. Reinforcement Learning Problems

In all our experiments we considered reinforcement learning problems. Tasks of this type usually require many trials and sophisticated algorithms in order to produce a well-performing agent, since a teacher signal is only available in the form of a scalar quantity, the reward. To the worse, a reward does not arrive at every time step, but is often given very sparsely and only for certain events. **Figure 2A** depicts a generic reinforcement learning loop. The agent observes the current state $s(t)$ of the environment and has to decide on an action $a(t)$. In particular, the agent samples an action according to policy $\pi(a|s)$, which is a probability distribution over actions a given a state s. Upon executing the action, the environment will advance to a new state $s(t+1)$ and the agent receives a reward $r(t)$. In all our experiments, the RL environment was simulated on the neuromorphic chip.

2.2.1. Markov Decision Process

Markov Decision Processes (MDPs) are a well-known and established model for decision making processes in literature. A MDP is defined by a five-tuple $(\mathbb{S}, \mathbb{A}, p, r, \gamma)$, with $\mathbb{S}$ representing the state space, $\mathbb{A}$ the action space, p the state transition function, r the reward function and γ a discount factor that weights future rewards differently from present ones. In particular, we are concerned here with such MDPs that exhibit discrete and finite state and action spaces. In addition, rewards are given in the range of $[0, 1]$. **Figure 2B** shows a simple example of such a MDP with $\|\mathbb{A}\| = 2$ and $\|\mathbb{S}\| = 3$.

The goal of solving a MDP is to find a policy actions that yields the largest discounted cumulative reward R that is defined as:

$$R = \sum_{t=0}^{T} \gamma^t r(t) \qquad (2)$$

In order to perform well on MDPs, the agent has to keep track of the rewarding transitions and must therefore represent the transition probabilities. Furthermore, the agent has to make a trade-off between exploring new transitions and consolidating already known transitions. Such problems have been studied intensively in literature and a mathematical framework was developed to optimally solve them by Bellman et al. (1954). The so-called Value-Iteration (VI) algorithm emerged from this framework and yields an optimal policy. Therefore, this algorithm is considered as the optimal baseline in all following MDP results.

In order to apply the L2L scheme, we introduce a family of tasks consisting of MDPs with a fixed size of the action and the state space. MDPs of that family are generated according to the following sampling procedure: whenever a new task is required, the rewards r and the transition probabilities p are randomly sampled from the range $[0, 1]$. In addition, the elements of p are normalized such that the outgoing probabilities for all actions in each state sum up to 1.

We report our results in the form of a normalized discounted cumulative reward, where we scale between the performance of a random action selection and the performance of an optimal action selection, given by a policy produced by VI.

2.2.2. Multi-Armed Bandits

As a second category of RL problems, we consider multi-armed bandit (MAB) problems. A MAB is best described as a collection

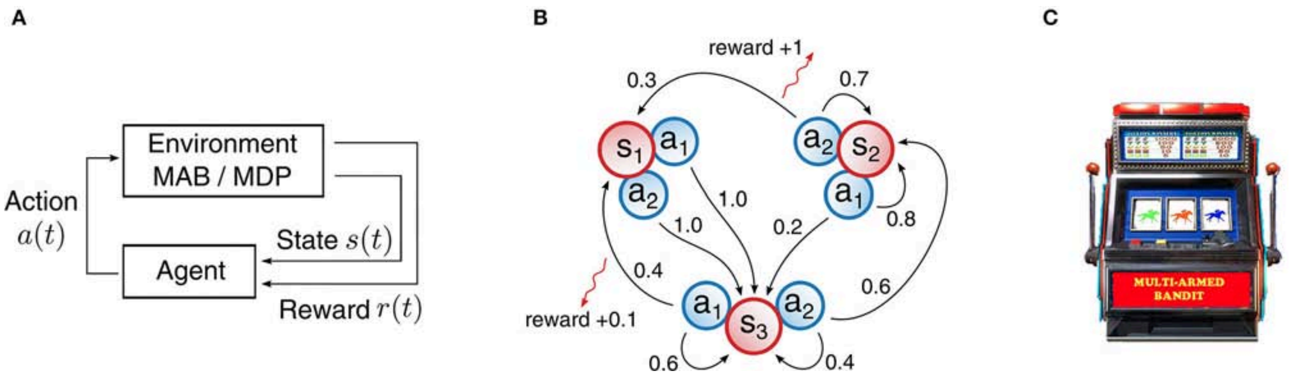


FIGURE 2 | Structure of reinforcement learning problems with examples for Markov Decision Processes (MDPs) and Multi-armed bandits (MABs). **(A)** General structure for reinforcement learning problems. An agent interacts with an environment in a loop. The agent selects an action based on state observations and receives a reward. **(B)** Example MDP with three states and two actions. State transitions are marked as arrows with annotated transition probabilities. A reward for a particular transition is indicated by a red arrow with the reward value along. Transitions with a probability of 0 or rewards with a value of 0 are omitted. **(C)** Illustration of a MAB: bandit arms can be pulled that produce a reward stochastically. "Multi-Armed Bandit" by Carson Reynolds is licensed under CC BY-SA 3.0.

of several one-armed bandits, each of which produces a reward stochastically when pulled. A depiction of which can be found in **Figure 2C**. In other words, one can view MAB problems as MDPs with a single state and multiple actions. Despite the deceptive simplicity of such problems, a great deal of effort was made in science to study these problems and the celebrated result of Gittins and Gittins (1979) showed that a learning strategy exists.

For the sake of brevity, we use the same notations for MABs as for MDPs. In particular, we say that the environment is always in one state s_1 and the agent is given the opportunity to pull several bandit arms i, which corresponds to actions a_i. In all experiments regarding MABs, we considered two-armed bandits, where each bandit produces a reward of either 0 or 1 with a fixed reward probability p_i. We investigate the impact of L2L on the basis of two different families of MAB tasks:

1. *unstructured bandits*: A task of this family is generated by sampling each of the two reward probabilities p_1, p_2 independently and uniform in $[0, 1]$.
2. *structured bandits*: A task of this family is generated by sampling the reward probability p_1 uniformly in $[0, 1]$ and compute $p_2 = 1 - p_1$.

Similar to MDPs, we report our results for MABs in the form of a normalized cumulative reward, where we scale between the performance of a random action selection and the performance of an oracle that always picks the best possible bandit arm. As a comparison baseline, we employ the Gittins index policy and note that the computation of the Gittins index value is calculated in the same way for both families. In particular, the Gittins index values are calculated assuming that the reward probabilities are independent *(unstructured bandits)*.

2.3. Neuromorphic Hardware - HICANN DLSv2

Various approaches for specialized hardware systems implementing spiking neural networks emerged and fundamentally differ in their realizations, ranging from pure digital over pure analog solutions using optical fibers up to mixed-signal devices (Indiveri et al., 2011; Nawrocki et al., 2016; Schuman et al., 2017). Every NM hardware comes with certain advantages and limitations, one promising platform is the HICANN-DLS (Friedmann et al., 2017), herein it is used in the prototype version 2.

The hardware is a prototype of the second generation BrainScaleS-2 system currently under development as part of the Human Brain Project neuromorphic platform (Markram et al., 2011). It represents a scaled-down version of the future full-size chip and is used to evaluate and demonstrate new features as illustrated in this work.

Conceptually the chip is a mixed-signal design with analog circuits for neurons and synapses, spike-based, continuous time communication and an embedded microprocessor. The NM hardware is realized in a 65 nm CMOS process node by the company TSMC. It features 32 neurons of the leaky-integrate-and-fire (LIF) type connected by a 32x32 crossbar array of synapses such that each neuron can receive inputs from a column of 32 synapses. Synaptic weights can be set with a precision of 6-bits and can be configured row-wise to deliver excitatory or inhibitory inputs. Synapses feature local short-term (STP) and long-term (STDP) plasticity, which is implemented by the embedded microprocessor described later. All analog time constants are scaled down by a factor of 1000 to represent an accelerated neuromorphic system compared to biological time-scales, a feature that is strongly exploited in this paper.

The embedded microprocessor is a 32-bit CPU implementing the Power-PC instruction set with custom vector extensions. It is used as a plasticity processing unit (PPU) to implement all synaptic weight changes. In particular, the PPU allows to devote memory to synapses in order to equip them with tagging mechanisms such as eligibility traces. As a general purpose processor, it can also act on any other on-chip data like neuron and synapse parameters as well as on the network connectivity. It can also send and receive off-chip signals like rewards or other control signals. Because of the large freedom in specifying programs for the PPU (written in C), we investigated different learning algorithms that are explained in section 2.5. They all exploit the proposed network structure from section 2.4 and have the commonality, that the reward information of the state transitions is encoded in the synaptic efficacy. In addition to learning algorithms, the plasticity processing unit also allows implementing environments for an agent. Since the system features a high speedup factor, any environment must also provide the same speedup factor in order to unlock full potential of the neuromorphic hardware, when using a closed-loop setup.

Some of the basic design rationales behind the second generation BrainScaleS-2 system with special emphasis on the PPU are described in Friedmann et al. (2017). **Figure 3A** shows the micrograph of the hardware and **Figure 3B** shows the measurement setup. In addition to other components, the measurement setup hosts the neuromorphic chip, a USB-Interface to connect the baseboard with a host computer as well as a separate FPGA board to control the experiments. The micrograph of the neuromorphic chip shows the different components and where they are located. A description of the actual prototype used in this work including details on the neuron implementation and the synaptic array can be found in Aamir et al. (2016).

2.4. Network Structure and Action Selection

As discussed in section 2.2, the agent is required to select an appropriate action $a(t)$ given a particular state $s(t)$ of the environment. We discuss in this Section how the agent can be implemented using a network of spiking neurons on neuromorphic hardware. Since our experiments were concerned with either Multi-armed bandits or Markov Decision Processes, we designed the network structure for the more general MDP problems. In particular, the design is based on the Markov Property of MDPs, using the fact that the next state $s(t + 1)$ solely depends on the chosen action $a(t)$ and the current state $s(t)$, similarly to Friedrich and Lengyel (2016).

Concretely, we make use of a feed-forward network of spiking neurons with two populations, as illustrated in **Figure 4A**. One population encodes the state of the environment (state population, marked in red) and the second population encodes all possible action choices (action population, marked in blue). We assume that all states exhibit the same number of possible actions. Under this assumption, the resulting agent commits to specific actions by the following action selection protocol: Given that the agent finds itself in state s_j, then the corresponding state neuron receives stimulating input and produces output spikes that are transmitted to the neurons a_i of the action population by excitatory synapses w_{ij}. Eventually, this stimulation will trigger

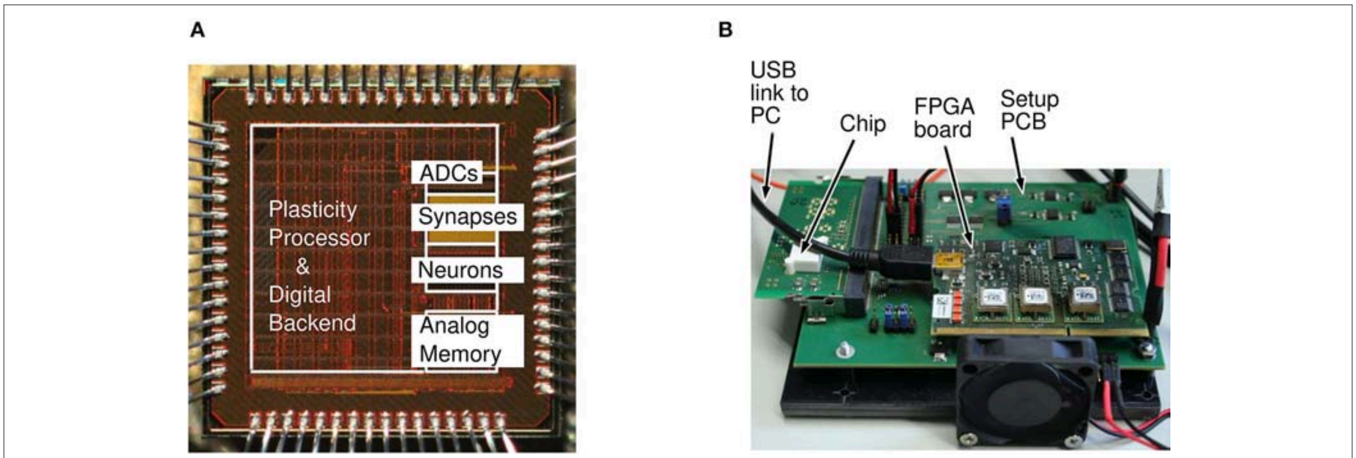


FIGURE 3 | Neuromorphic chip micrograph and measurement setup adopted from Aamir et al. (2016). **(A)** Micrograph of the neuromorphic hardware. The plasticity processing unit, the area responsible for the synaptic part, the neuronal part, a memory area as well as analog to digital converters (ADCs) are marked. **(B)** Measurement setup and prototype board. The board shows the neuromorphic chip itself, the interface to the host computer and a supportive FPGA board.

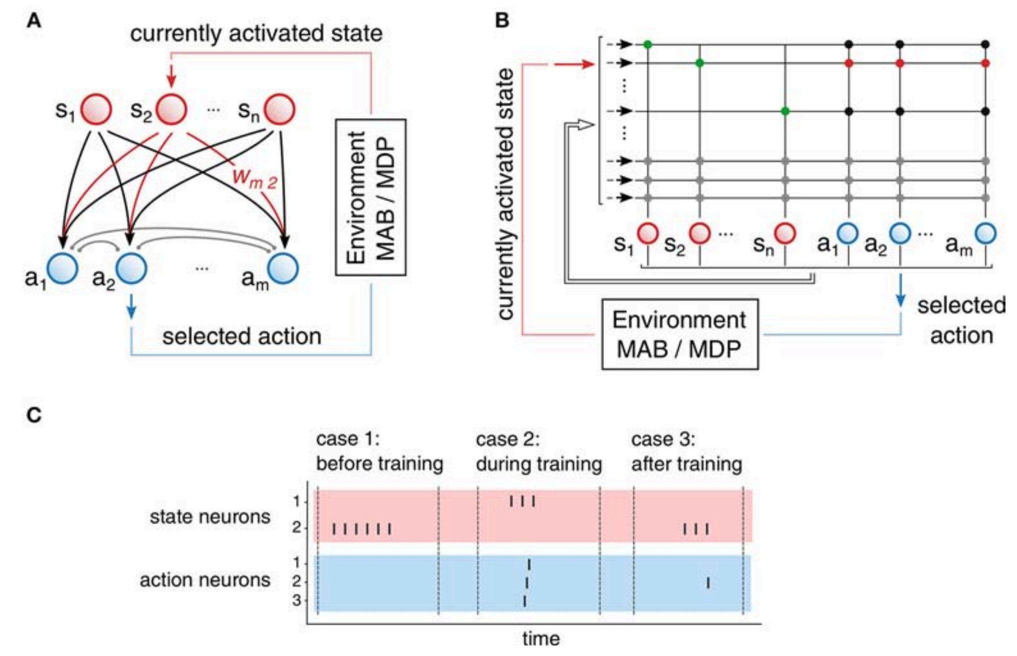


FIGURE 4 | Neural network structure and realization on neuromorphic hardware. **(A)** Network structure with two populations: state population (red), action population (blue). Excitatory synapses w_{ij} (black and red) are plastic and used for learning. Inhibitory synapses (gray) introduce mutual inhibition in the action population. **(B)** Mapping of the network onto the neuromorphic hardware. Synapses are organized in crossbar array of size (32 x 32). We use autapses (green) for persistent excication of state neurons. Persistent excitation is stopped by additional inhibitory synapses that connect the action population to the state population. **(C)** Three examples of the action selection process. In case 1, none of the action neurons received enough input to emit a spike: a random action is selected. In case 2, each action neuron emits a spike: A random action among active neurons is selected. In case 3, only a single neuron of the action population emits a spike that determines the selected action.

a spike in the action population, depending on the synaptic strengths w_{ij}. The action $a(t)$ that will be taken is determined by the neuron of the action population that emits a spike first. In addition, neurons coding for actions are connected inhibitory among each other with synapses of strength ξ, through which a WTA-like network structure arises. Due to this mutual inhibition, mostly a single neuron of the action population will emit a spike and hence, trigger the corresponding action.

In practice, additional tricks are required to implement the proposed scheme on the neuromorphic device, see **Figure 4B**. To continually excite the active state neuron, we send a single spike that triggers a persistent firing through strong excitatory autapses (marked in green). If a neuron from the action population eventually emits a spike, the active state neuron needs to be prevented from further spiking. For this purpose, we use inhibitory synapses of strength ζ projecting from action neurons to state neurons. Due to synaptic delays, more than one action neuron may emit a spike. In such a case, an action is randomly selected among the set of active neurons. It is to be noted that smaller inhibition weights lead to more random exploration, because insufficient inhibition will not prevent spikes of other action neurons, in which case action selection becomes randomized.

One other implementation detail comes from the fact that the synaptic weights on the NM hardware yield a limited resolution of only 6 bit. This might cause that weights saturate at either 0 or the maximum weight value and prevent efficient learning. To avoid this problem, the weights w_{ij} are rescaled with a certain frequency f_{rescale} according to:

$$k = \frac{W_{\max} - W_{\min}}{\max(w_{ij}) - \min(w_{ij})} \tag{3}$$

$$d = W_{\max} - k\max(w_{ij}) \tag{4}$$

$$w'_{ij} = kw_{ij} + d \tag{5}$$

where $W_{\max}$ and $W_{\min}$ provide the upper and lower rescale boundary.

Figure 4C depicts typical examples of the action selection process for three common cases occurring throughout the learning process. In case 1 (usually before training), a state neuron, i.e., corresponding to state 2, is active and persistently emits a spike. However, none of the synapses connecting to the action neurons is strong enough to cause a spike. In such a case, after a predefined time, the state neuron is externally inhibited and a random action is selected by the implementation of the environment. In case 2 (likely during learning), another state neuron is active, but all synapses to the action neurons are strong enough to cause every action neuron to spike before the mutual inhibition sets in. In such a case, a random action among the active action neurons is selected (random selection is performed by the environment). Eventually, the system reaches case 3 (after learning), where a single action neuron is excited by a given state neuron.

Learning in this network structure is implemented by synaptic plasticity rules that act upon the excitatory weights w_{ij} projecting from the state to the action population. In particular, these weights pin down which action has the highest priority for each state.

2.5. Learning Algorithms

2.5.1. Q-Learning

MDPs have been studied intensively in computer science and a rigorous framework on how to solve problems of this kind optimally was introduced by Bellman. An important quantity in MDPs is the so-called Q-Function, or Action-Value function. The Q-Function $Q^{\pi}(s,a)$ expresses the expected discounted cumulative reward, when the agent starts in state s, takes action a and subsequently proceeds according to its policy π. Formally, one writes this as:

$$Q^{\pi}(s_j, a_i) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(t+k+1) | s(t) = s_j, a(t) = a_i\right] \tag{6}$$

where γ is the discount factor of the MDP and $r(t)$ is the immediate reward at time step t. As discussed before in section 2.2.1, we consider only discrete MDPs and the Q-Functions can therefore be represented in a tabular form. This property suits our network structure, since the synapses that project from the state population to the action population w_{ij} can represent all Q-values $Q^{\pi}(s_j, a_i)$. Hence, we define $w_{ij} \overset{def}{=} Q^{\pi}(s_j, a_i)$.

To solve MDPs, the goal is to determine the optimal policy π^*. A common approach is to infer the Q-Function of an optimal policy Q^* and then reconstruct the policy according to:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} Q^*(s, a') \\ 0 & \text{else} \end{cases} \tag{7}$$

Indeed, as we aim to encode Q-values in synaptic weights w_{ij}, we emphasize that the argmax operation will be naturally carried out by the spiking neural network, as proposed in section 2.4. To infer the Q-values of the optimal policy, we derive rules of synaptic plasticity based on temporal difference algorithms as proposed by Sutton and Barto (1998).

2.5.1.1. TD(1)-Learning

Temporal Difference Learning (TD(1)-Learning) was developed as a method to obtain the optimal policy. The estimate of the optimal Q-Function is improved based on single interactions with the environment and TD(1)-Learning is guaranteed to converge to the correct solution (Watkins and Dayan, 1992; Dayan and Sejnowski, 1994). Based on TD(1), the synaptic weight updates take on the following form:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\left(r(t) + \gamma \max_k w_{kj}(t) - w_{ij}(t)\right)$$
$$\text{for } s(t) = s_j, a(t) = a_i \tag{8}$$

Where α denotes a learning rate.

2.5.1.2. TD(λ)-Learning

The convergence speed of TD(1)-Learning can be further improved if one uses additional eligibility traces $e_{ij}(t)$ per synapse. The resulting algorithm is then referred to as TD(λ)-Learning. In particular, the trace e_{ij} indicates to what extent a current reward makes the earlier visited state-action pair (s_j, a_i) more valuable and several convergence proofs of the resulting algorithm have been established (Dayan, 1992; Dayan and Sejnowski, 1994). To implement the algorithm, we update eligibility traces at every time step t according to the schedule

$$e_{ij}(t) = \begin{cases} \gamma\lambda e_{ij}(t-1) + 1 & \text{if } s(t) = s_j \text{ and } a(t) = a_i \\ \gamma\lambda e_{ij}(t-1) & \text{otherwise} \end{cases} \tag{9}$$

where the parameter $\lambda \in [0, 1]$ controls how many state transitions are taken into account. In the limit of $\lambda = 1$ one obtains TD(1)-Learning. In addition, we define an error $\delta(t)$ according to

$$\delta(t) = r(t) + \gamma \max_k w_{kj}(t) - w_{ij}(t) \quad \text{for } s(t) = s_j, a(t) = a_i \tag{10}$$

which enables us to express the resulting plasticity rule as a product of the eligibility trace and error $\delta(t)$. This update is carried out for every synapse w_{ij}:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta(t)e_{ij}(t) \quad \text{for all } i, j \tag{11}$$

2.5.2. Meta-Plasticity

In order to tailor the specific update rule toward the actual task family at hand, we approached the problem also from the perspective of meta-plasticity. That is, we represent the synaptic weight update by a parameterized function approximator. We then optimize its parameters with L2L in such a way that a useful learning rule for a given task family emerges. We used a multilayer perceptron, the architecture of which is visualized in **Figure 5**. The perceptron receives five inputs, computes seven hidden units with sigmoidal activation and provides one output, the weight update Δw_{ij}. Effectively, the input to output mapping of this approximator is specified by a number of free parameters θ (weights of the multilayer perceptron) that are considered as hyperparameters and optimized as part of the L2L procedure. Since the multilayer perceptron is a type of an artificial neural network, this plasticity rule is referred to as ANN learning rule. The update of synaptic weights w_{ij} thus takes on the general form of:

$$w_{ij}(t+1) = w_{ij}(t) + f_{\text{ANN}}(\text{inputs}_{ij}(t); \theta) \tag{12}$$

The specific choice of inputs is salient for the possible set of learning rules that can emerge. In the case of the ANN learning rule, we only considered structured MAB, where each of the two synapses is updated at every time step. We set the inputs in this case to a vector

$$\text{inputs}_{i\,1}(t) = \begin{pmatrix} t \\ \mathbb{1}_{a(t)=a_i} \\ r(t) \\ w_{i\,1}(t) \\ w_{3-i\,1}(t) \end{pmatrix} \tag{13}$$

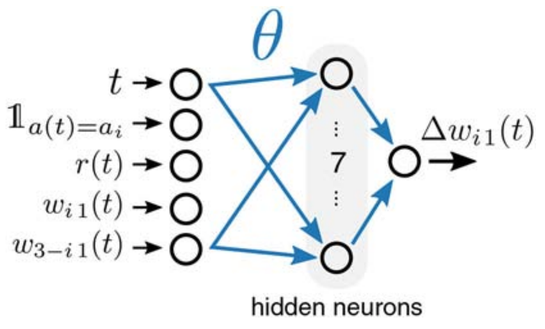


FIGURE 5 | Meta-plasticity for a two-armed bandit task. The plasticity rule is represented by a parametrized multi-layer perceptron with one hidden layer (denoted as ANN). It receives as inputs the time step t, a binary flag $\mathbb{1}_{a(t)=a_i}$ that indicates if the weight to be updated was responsible for the selected action, the obtained reward $r(t)$, as well as the weights $w_{i\,1}$ and $w_{3-i\,1}$.

that is composed of the current time step t, the obtained reward $r(t)$, the weight $w_{i\,1}(t)$, and the weight of the synapse associated to the other bandit arm $w_{3-i\,1}(t)$. In addition, we included here a binary flag $\mathbb{1}_{a(t)=a_i}$ that is one iff the postsynaptic neuron caused the executed action at the last time step.

2.6. Analysis of Meta-Plasticity

After optimizing an artificial neural network in our meta-plasticity approach, we may have limited insight in what causes the emergent plasticity rule to work well. Therefore, we conduct in section 3.3 an analysis of the arising plasticity rule based on an approach called functional Analysis of Variance (fANOVA) which was presented by Hutter et al. (2014). This method originally aims to assess the importance of hyperparameters in the machine learning domain. It does so by fitting a random forest to the performance data of the machine learning model that was gathered using different hyperparameters.

We adopted this method but applied it to a slightly different, but related problem. Our goal is to assess the impact of each input of the ANN rule with respect to its output. To do so, the weights Θ of the plasticity network remain fixed, while the input values to the plasticity network as well as the output from the plasticity network are considered as inputs to the fANOVA framework. Based on this data, a random forest with 30 trees is fitted and the fraction of the explained variance of the output with respect to each input variable can be obtained.

3. RESULTS

This section presents the results of our approach implemented on the described neuromorphic hardware. First, we report how L2L can improve the performance and learning speed in section 3.1. Then, we investigate the impact of outer loop optimization algorithms in section 3.2 and demonstrate in section 3.3 that Meta-Plasticity yields competitive performance, while also enhancing transfer learning capabilities. Finally, we investigate the speedup gained from the neuromorphic hardware by comparing our implementation on the NM hardware to a pure software implementation of the same model in section 3.4.

3.1. Learning-to-Learn Improves Learning Speed and Performance

Here, we first demonstrate the generality of our network structure when applied to Markov Decision Processes. Then, we examine the effects of an imposed task structure more closely by investigating Multi-armed Bandit problems. To efficiently train the network of spiking neurons, we employed Q-Learning and derived corresponding plasticity rules, as described in see section 2.5.1. The plasticity rule, as well as the concrete implementation on NM hardware, are influenced by hyperparameters Θ that we optimized by L2L, such that the cumulative discounted reward for a given family of tasks is improved on average, see section 2.1.

We implemented a neuromorphic agent that learns MDPs. In fact, the proposed network structure in section 2.4 is particularly designed for such tasks and we applied concretely TD(λ), see Equation (11). Hyperparameters included all occurring parameters of the employed TD(λ)-Learning rule α, γ, λ, the inhibition strength among the action neurons ξ, the strength of inhibitory weights connecting the action neurons to the state neurons ζ, as well as the variables influencing the hardware-specific rescaling $f_{\text{rescale}}, W_{\max}$ and $W_{\min}$. Therefore, the complete hyperparameter vector was given as $\Theta = (\alpha, \gamma, \lambda, \xi, \zeta, f_{\text{rescale}}, W_{\max}, W_{\min})$. We used the discounted cumulative reward, Equation (2), as the fitness function $f(C; \Theta)$ and optimized Θ using CE. We used a batch size of $N = 20$.

The results for the MDP tasks are depicted in **Figure 6A** where we report the discounted cumulative reward for $T = 2,000$ steps. The discounted cumulative reward is normalized in such a way, that VI is scaled to 1 and the random policy is scaled to 0. To compare with, we used TD(λ)-Learning as a baseline, using the implementation from a software library[1] without a spiking neural network (green line).

We found that applying L2L improved the discounted cumulative reward (red solid line), compared to the case where the hyperparameters are randomly chosen (blue line). In addition, the learning speed was also increased, which can be seen in the zoom depicted in **Figure 6B**.

In the case of MABs, we focused on small networks and two arms in the bandit, which allowed us to complement the results that were obtained for general MDPs of larger size. We considered two families of MABs: *unstructured bandits* and *structured bandits* (2.2.2) which the neuromorphic agent had to learn using the TD(1)-Learning rule, see Equation (8), where we set $\gamma = 1$. In addition, we introduced here a learning rate schedule $\alpha(t) = \alpha_{\text{decay}}^t \cdot \alpha_0$ that decays a base learning rate α_0 at every time step by a constant decay factor of $\alpha_{\text{decay}} \in [0, 1]$. We then used L2L to carry out a hyperparameter optimization separately for both MAB families and optimized the parameters of the TD(1)-Learning rule α_0 and α_{decay}, the inhibition strength among action neurons ξ and the inhibitory

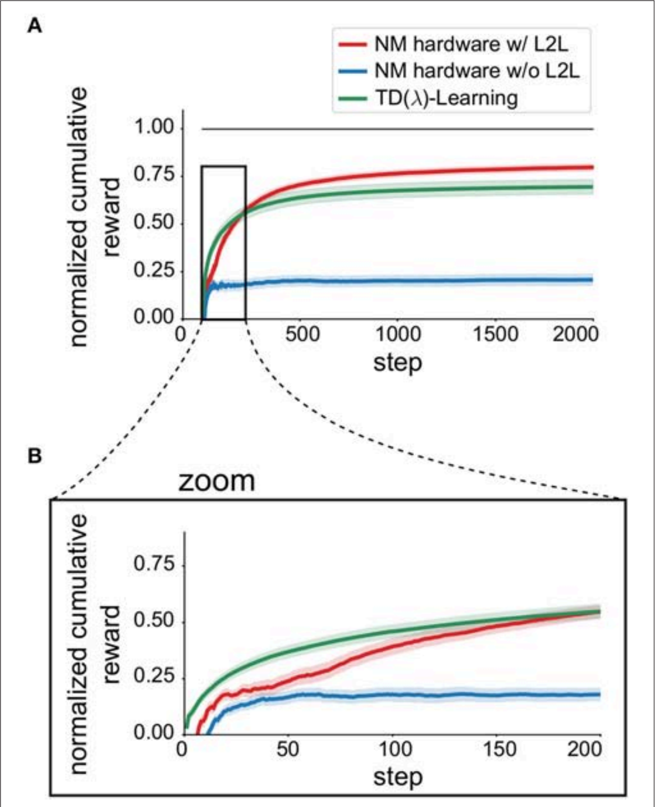


FIGURE 6 | Impact of L2L for Markov Decision processes. **(A)** Average learning performance on the MDP task family ($\|\mathbb{S}\| = 2$, $\|\mathbb{A}\| = 4$) using TD(λ)-Learning, see Equation (11). Learning performance is expressed as the normalized cumulative discounted reward (0 random, 1 optimal) and is averaged over 50 different tasks. Shaded areas mark the uncertainty of the mean. **(B)** Zoom into the first 200 steps to emphasize increased learning speed.

weights of synapses that connect the action population to the state population ζ. Hence, the hyperparameter vector was given as $\Theta = (\alpha_0, \alpha_{\text{decay}}, \xi, \zeta)$. We used the cumulative reward as the fitness function $f(C; \Theta)$ and optimized Θ using CE. We used a batch size of $N = 40$.

In **Figure 7** we report the performance results that were obtained before and after applying L2L. The agent interacted for $T = 100$ steps with a single MAB and we compare with a baseline given by the Gittins index policy, as described in section 2.2.2. We found that after performing a L2L optimization the performance was enhanced, which was even more apparent for *structured bandits*. In particular, L2L endowed the agent with a better learning speed, which is exhibited by a faster rising of the performance curve. This can only be achieved when the hyperparameters of the learning system are well-tailored to the tasks that are likely to be encountered, which was the responsibility of L2L. We also observed that the agent could still learn a MAB task to a reasonable level even if no L2L optimization was carried out. This is implied by the fact that TD(1)-Learning is primed to learn RL tasks. However, this also raises the question of how well such a general plasticity rule can

[1]https://pymdptoolbox.readthedocs.io/en/latest/index.html

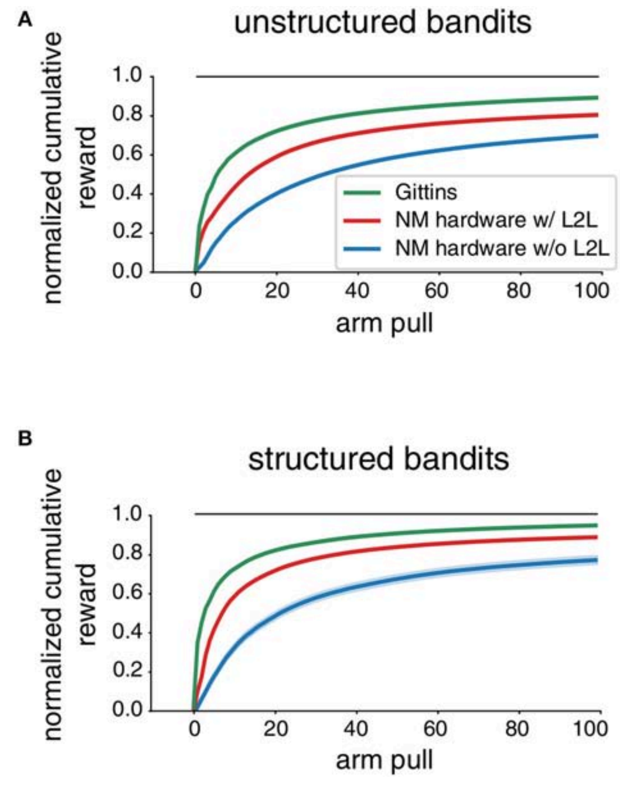


FIGURE 7 | Impact of L2L for Multi-armed Bandit tasks. We report the normalized cumulative reward (0 random, 1 optimal) on MAB tasks averaged over 1, 000 different tasks. The shaded areas mark the uncertainty of the mean. The neuromorphic agent uses TD(1)-Learning, see Equation (8). **(A)** *unstructured bandits* **(B)** *structured bandits*.

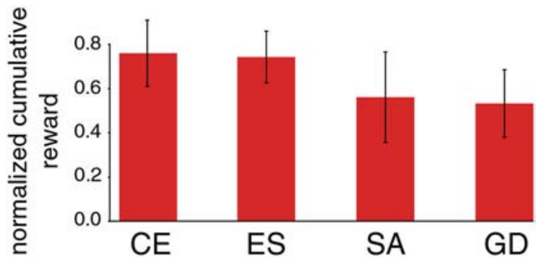


FIGURE 8 | Performance impact of different outer loop algorithms. We exhibit the performance of an L2L optimized neuromorphic agent on MDP tasks with $\|\mathbb{S}\| = 2$ and $\|\mathbb{A}\| = 4$. The performance is measured as the final normalized discounted cumulative reward after $T = 2,000$ steps and are averaged over 50 different tasks. We compare Cross-Entropy (CE), Evolution strategies (ES), Simulated annealing (SA), and numerical Gradient descent (GD), as described in section 2.1. The dimensionality of the hyperparameter vector was 8, as in section 2.2.1.

adapt to the level of variations exhibited by analog circuitry. We consider extensions in section 3.3.

3.2. Performance Comparison of Gradient-Free Optimization Algorithms in the Outer Loop

The results presented so far suggest that the concept of L2L can improve the overall performance and also lays the foundation that abstract knowledge about the task family at hand is integrated into an agent. However, the choice of a proper outer loop optimization algorithm is also crucial for this scheme to work well. The modular structure of the L2L approach used in this paper allows to interchange different types of optimization algorithms in the outer loop for the same inner loop task. To demonstrate the impact in terms of performance when using different optimization algorithms, several such algorithms were investigated for both general MDPs and also for specialized MAB tasks. **Figure 8** shows a comparison of the final discounted cumulative reward at the end of the tasks for different outer loop optimization algorithms.

Depending on the inner loop task considered, we found that the cross-entropy (CE) method, as well as evolution strategies (ES), work well because both aim to find a region in the hyperparameter space, where the fitness is high. This property is particularly desired when it comes to noise in the fitness landscape due to imperfections of an underlying neuromorphic hardware. In addition, both can cope with noisy fitness evaluations and do not overestimate a single fitness evaluation which could easily lead to a wrong direction in the presence of high noise in the fitness landscape.

However, a simpler algorithm such as simulated annealing (SA) can also find a hyperparameter set with rather high fitness. Especially when running multiple separate annealing processes in parallel with different starting points, the results can almost compete with the ones found by CE or ES. However, SA does not aim at finding a good parameter region but just tries to find a single good set of working hyperparameters. This is prone to cause problems because a single good set of hyperparameters offers less robustness compared to an entire region of well-performing hyperparameters. A simple numerical gradient-based approach did not yield good results at all because of the noisy fitness landscape. In general, the developer is free to choose any optimization algorithm in the outer loop when using L2L. New algorithms can also be implemented which are specially tailored to a particular problem class, which can lead to a new research direction.

3.3. Performance Improvement Through Meta-Plasticity

Since the plasticity rule used so far is based on TD(1)-Learning, and also agnostic to the hardware being used, we raised the question if one could improve training on particular tasks by using an evolved plasticity rule, tailored specifically toward the neuromorphic device and task family at hand. We specified the plasticity rule by a multilayer perceptron with 7 hidden units (**Figure 9**) and considered the weights thereof as hyperparameters. This is apparently the first example of meta-plasticity on neuromorphic hardware, where a rule for synaptic plasticity is evolved through optimization by L2L.

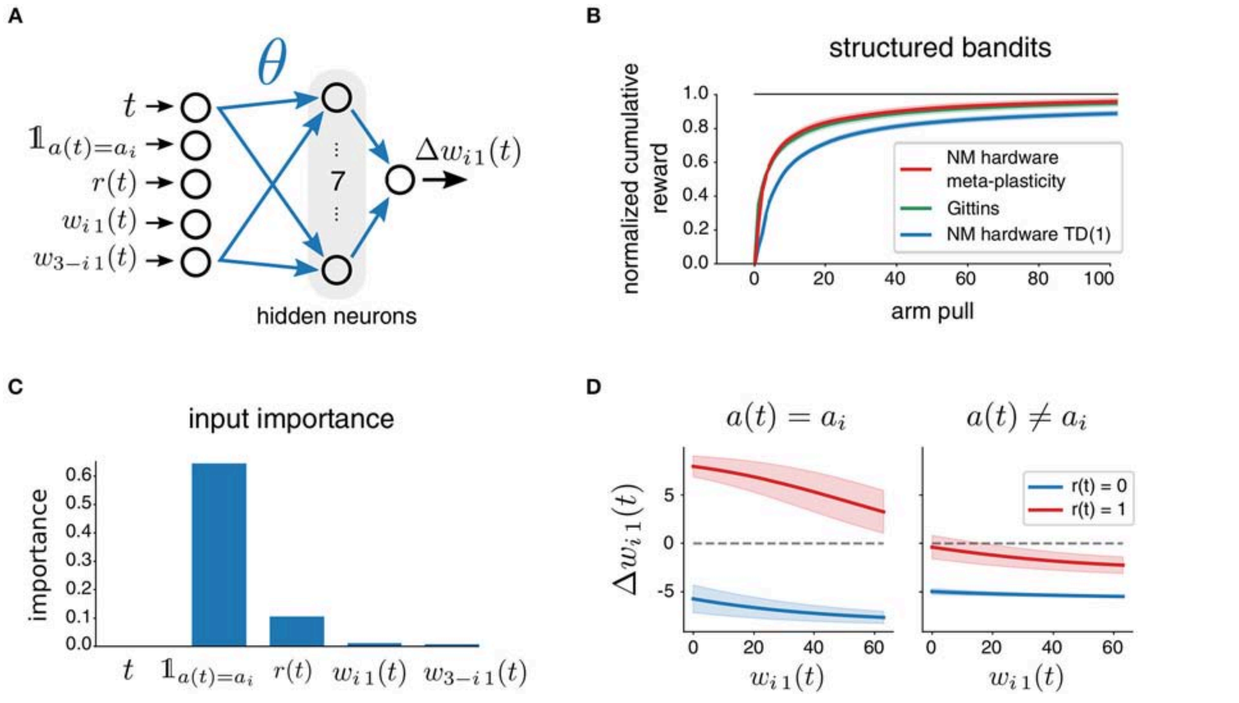


FIGURE 9 | Meta-plasticity for a two-armed bandit task. **(A)** The plasticity rule is represented by a parametrized multi-layer perceptron. **(B)** Performance of the meta-plasticity as compared to an optimized TD(1)-Learning neuromorphic agent and Gittins index on structured MABs. **(C)** Relative contributions of each input to the variance of the weight update as computed by fANOVA. Mostly responsible are the action flag and the reward signal. **(D)** The weight update as shown for the different possible cases of $\mathbb{1}_{a(t)=a_i}$ and $r(t)$ depending on the current weight $w_{i\,1}$. Shaded areas indicate effects of inputs that are not fixed by the variables on the axes: t and $w_{3-i\,1}$.

To test the approach, we used L2L to optimize all occurring hyperparameters on the task family of *structured bandits*. In particular, the hyperparameter vector was composed of the parameters of the plasticity rule θ and the inhibition strengths ξ and ζ: $\Theta = (\theta, \xi, \zeta)$. We used the cumulative reward, Equation (2), as the fitness function $f(C; \Theta)$ and optimized Θ using CE with a batch size of $N = 40$.

We summarize our results in **Figure 9** and observed a drastic increase in learning performance. Clearly, the use of meta-plasticity endowed the agent with better skill at learning *structured bandits*, as compared to the TD(1)-Learning rule. It also allows the agent to achieve a performance that is on the same level as the Gittins index policy. This highlights that the evolved plasticity rule can absorb task-structure, and counteract possible negative effects of imperfections in the neuromorphic hardware.

Even though the arising learning rule performs well on average on the family of tasks it has been trained on, there is no theoretical guarantee for that. Hence, an analysis of the optimized learning rule was conducted, where we examined the importance of the multiple inputs provided to the update rule for the resulting output, see **Figure 9C**. Apparently, the most important inputs are the flag that represents if the current weight was responsible for the last action and the obtained reward. Since both of the inputs can assume only two values, one can visualize the four different cases in four different curves. We report the expected weight change depending on the current weight, see **Figure 9D**, where we average over other unspecified inputs. Updates for weights which were responsible for the previous action are in the direction of the obtained rewards. Hence, the meta-plasticity rule reinforced actions depending on the reward outcome, similarly to Q-learning rules. Interestingly however, the update of the synaptic weight which had not caused the last action was always negative independently of the reward. We believe that L2L simply found that it does not matter what happens to the weight that did not cause actions, because as long as it does not increase, it will not disturb the current belief of the best bandit arm.

To test if the reinforcement learning agent on the neuromorphic hardware has been optimized for a particular range of tasks, we carried out another experiment. We tried to answer if the agent can take advantage of the abstract task structure if it was present. To do so, we always tested learning performance on *structured bandits*, denoted as $\mathcal{F}'$. For optimization with L2L, we instead used either *unstructured bandits* or *structured bandits*, and we denote the family on which hyperparameter optimization was carried out by $\mathcal{F}$. This experimental protocol (**Figure 10A**) allowed us to determine to which extent abstract task structure can be encoded in hyperparameters. We report the results for neuromorphic agents in **Figure 10B**, where we considered the TD(1)-Learning rule and the meta-plasticity learning rule. Consistently, we observed that optimizing

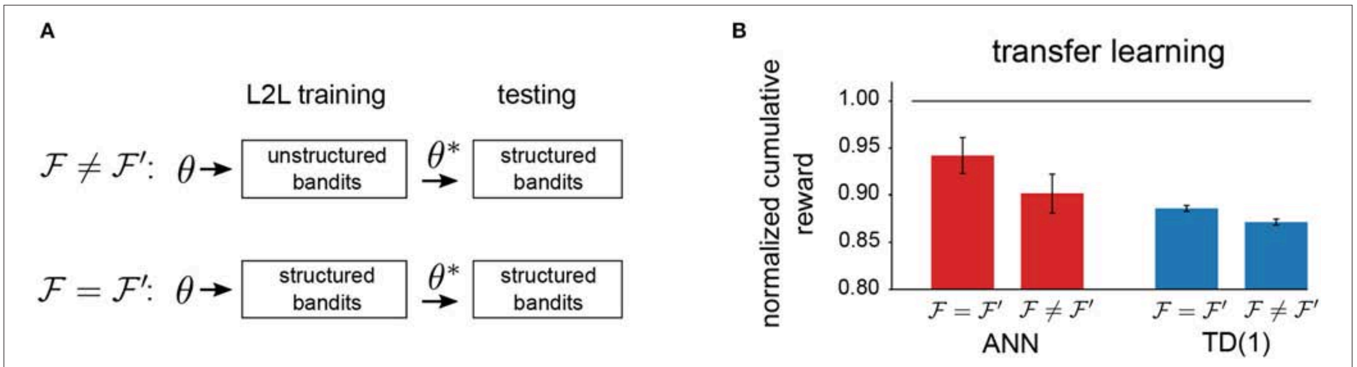


FIGURE 10 | Transfer-learning. **(A)** Setup to investigate transfer learning capabilities. **(B)** Comparison of transfer-learning capabilities of different learning rules. The meta-plasticity update rule (ANN) can potentially encode more task structure through a greater dimensionality of the hyperparameter vector Θ.

hyperparameters for the appropriate task family enhances performance. However, we conjecture that the greater adjustability of the meta-plasticity learning rule renders it to be better suited for transfer learning as compared to TD(1)-Learning rule.

3.4. Exploiting the Benefit of Accelerated Hardware for L2L

One of the main features of neuromorphic hardware devices is the ability to simulate spiking neural networks very fast and efficiently. To make this more explicit for the MDP tasks, a software implementation with the same network structure and the same plasticity rule was conducted on a standard desktop PC using one single core of an Intel^{TM} Xeon^{TM} CPU X5690 running at 3.47 GHz. The spiking neural network was implemented using the Neural Simulation Tool (NEST) (Gewaltig and Diesmann, 2007) with a Python interface and the plasticity rule as well as the environment were also implemented in Python. To have a better comparison, two families of MDP tasks with different sizes of $\|\mathbb{S}\|$ and $\|\mathbb{A}\|$ were defined. The first family is defined by $\|\mathbb{S}\| = 2$ and $\|\mathbb{A}\| = 4$ (small MDP) and the second family by $\|\mathbb{S}\| = 6$ and $\|\mathbb{A}\| = 8$ (large MDP).

Figure 11A shows a comparison of the simulation time needed for a single randomly selected MDP tasks, averaged over 50 MDPs and for each of the two families. The simulation times include implementation specific overheads, for example, the communication overhead with the neuromorphic hardware. One can see that the simulation time needed for MDP tasks with both sizes are shorter using the neuromorphic hardware and in addition, the simulation time needed to solve the larger task does not increase. First, this indicates, that the neuromorphic hardware can carry out the simulation of the spiking neural network faster and second, that using a larger network structure does not yield an additional cost, as long as the network can fit on the NM hardware. In contrast to this, using more neurons requires longer simulation times in pure software. A similar key message can be found in **Figure 11B**, where instead of a single MDP run, an entire L2L run is evaluated on the neuromorphic hardware as well as with the software implementation. Both, the L2L run on neuromorphic hardware as well as the one in software can in principle be easily parallelized when using more hardware systems or more CPU cores which would decrease the overall simulation time. Note that scheduler overheads are not taken into considerations.

4. DISCUSSION

Outstanding successes have been achieved in the field of deep learning, ranging from scientific theories and demonstrators to real-world applications. Despite impressive results, deep neural networks are not out of the box suitable for low-power or resource-limited applications. Instead, spiking neural networks are inspired by the brain, an arguably very power efficient computing machine. In this work we employ a neuromorphic hardware that was designed to port key aspects of the astounding properties of this biological circuitry to silicon devices.

The human brain has been prepared by a long evolutionary process with a set of hyperparameters and learning algorithms that can be used to cover a large variety of computing and learning tasks. Indeed, humans are able to generalize task concepts and port them to new, similar tasks, which provides them with a tremendous advantage as compared to most of the contemporary neural networks. In order to mimic this behavior, we employed gradient-free optimization techniques, such as the cross-entropy method or evolutionary strategies (see section 2.1), applied in a Learning-to-Learn setting. This two-looped scheme combines task-specific learning with a slower evolutionary-like process that results in a good set of hyperparameters as demonstrated in section 3.1. The approach is generic in the sense that both, the algorithms mimicking the slower evolutionary process and the learning agent can be exchanged. In principle, any agent with learning capabilities can be used as the learning agent and any optimization algorithms as the evolutionary process. We found that some outer loop optimization algorithm perform better than others and the optimization algorithms should ideally be chosen with the inner loop task in mind. Outer loop optimization algorithms need to operate in a high-dimensional parameter space, have the ability to deal with noisy result evaluations, have the ability to find a good final solution and also require a low number of parameter evaluations before

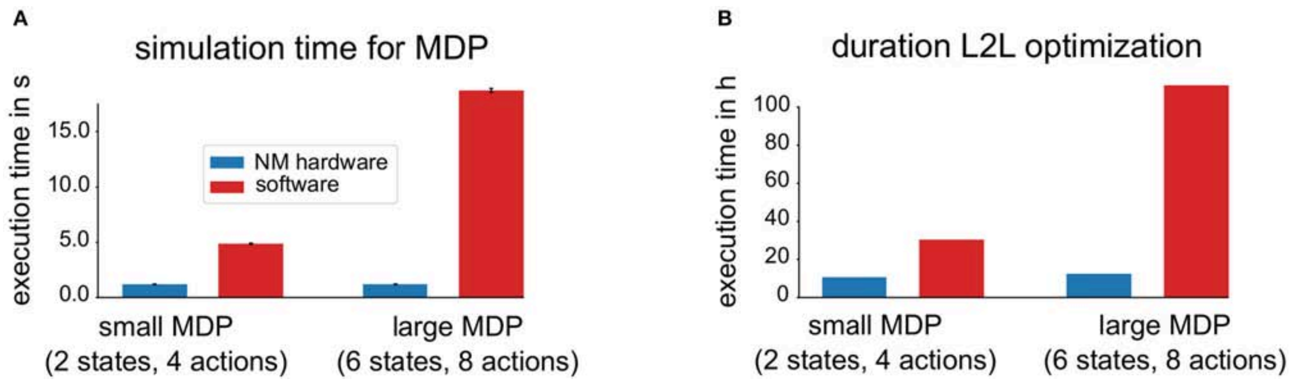


FIGURE 11 | Impact of accelerated neuromorphic hardware on simulation time. **(A)** Shows the comparison of the required simulation time, averaged over 50 different MDP tasks of two different sizes. In software simulations, only a single CPU core was used. The simulation time of the NM hardware is shorter and remains constant for the two families. **(B)** Shows the duration comparison for an entire L2L optimization for two different families.

reaching a good solution. Algorithms that aim to find a region of hyperparameters with high performance such as evolution strategies or cross-entropy worked the best for us, see section 3.2.

L2L offers both, either to find optimal hyperparameters for a fixed individual task or to boost transfer learning capabilities of an agent when using a family of tasks. In addition, new optimization algorithms can be developed to further improve performance in the outer loop of L2L. In this work, we used reinforcement learning problems in connection with NM hardware to demonstrate the aforementioned benefits.

In particular, the concept of L2L allows to shape highly adjustable plasticity rules for specific task families. The usage is not only limited to spiking neural networks but can also be applied to artificial neural networks. This may yield potential for a future research direction. Apparently, this is the first time that the idea of L2L and Meta-Plasticity was applied to a NM hardware, see section 3.3. In addition, the NM hardware provides the possibility to implement advanced plasticity rule on a separate digital processor on-chip. This enables the search for new plasticity rules and might also enable new research directions.

A central role in the approaches explained in this paper is the used NM hardware. It allows to emulate a spiking neural network with a significant speedup compared to the biological equivalent, which makes a large number of computations, required in the L2L scheme feasible. To quantify the overall speedup of the accelerated NM hardware, a comparison with a pure software simulation on a conventional computer was carried out (see **Figure 11**). We conclude that the two-looped L2L scheme as well as the highly adjustable on-chip plasticity rule are especially suited for accelerated neuromorphic hardware.

AUTHOR CONTRIBUTIONS

WM, TB, and FS developed the theory and experiments. TB implemented and conducted experiments with regard to MDPs, benchmarked performance impact of outer loop optimization algorithms and probed the performance benefit of NM hardware. FS implemented and conducted experiments with regard to MABs. FS, CP, and WM conceived meta-plasticity, FS and CP implemented it. FS tested the benefits in transfer learning. TB, FS, CP, WM, and KM wrote the paper.

FUNDING

This research/project was supported by the HBP Joint Platform, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

ACKNOWLEDGMENTS

We thank Anand Subramoney for his support and the contributions to the Learning-to-Learn framework. We are also grateful for the support during the experiments with the neuromorphic hardware. In particular, we like to thank David Stöckel, Benjamin Cramer, Aaron Leibfried, Timo Wunderlich, Yannik Stradmann, Christian Mauch, and Eric Müller. Furthermore, we also like to thank Elias Hajek for useful comments on earlier versions of the manuscript.

REFERENCES

Aamir, S. A., Muller, P., Hartel, A., Schemmel, J., and Meier, K. (2016). "A highly tunable 65-nm CMOS LIF neuron for a large scale neuromorphic system," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference* (IEEE), 71–74.

Aamir, S. A., Stradmann, Y., Müller, P., Pehle, C., Hartel, A., Gruebl, A., et al. (2018). An accelerated LIF neuronal network array for a large scale mixed-signal neuromorphic architecture. *IEEE Trans. Circuits Syst.* 12, 4299–4312. doi: 10.1109/TCSI.2018.2840718

Ambrogio, S., Narayanan, P., Tsai, H., Shelby, R. M., Boybat, I., di Nolfo, C., et al. (2018). Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature* 558, 60–67. doi: 10.1038/s41586-018-0180-5

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and Learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems 31,* eds. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc.), 787–797. Available online at: http://papers.nips.cc/paper/7359-long-short-term-memory-and-learning-to-learn-in-networks-of-spiking-neurons.pdf

Bellman, R., Glicksberg, I., and Gross, O. (1954). The theory of dynamic programming as applied to a smoothing problem. *J. Soc. Indus. Appl. Math.* 2, 82–88. doi: 10.1137/0102007

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P. (1992). The convergence of TD(X) for general X. *Mach. Learn.* 8, 341–362. doi: 10.1007/BF00992701

Dayan, P., and Sejnowski, T. J. (1994). TD(λ) converges with probability 1. *Mach. Learn.* 14, 295–301. doi: 10.1007/BF00993978

Finn, C., Abbeel, P., and Levine, S. (2017). "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research*, Vol. 70, eds. D. Precup and Y. Whye Teh (Sydney, NSW: International Convention Centre), 1126–1135.

Friedmann, S., Schemmel, J., Grübl, A., Hartel, A., Hock, M., and Meier, K. (2017). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Trans. Biomed. Circuits Syst.* 11, 128–142. doi: 10.1109/TBCAS.2016.2579164

Friedrich, J., and Lengyel, M. (2016). Goal-directed decision making with spiking neurons. *J. Neurosci.* 36, 1529–1546. doi: 10.1523/JNEUROSCI.2854-15.2016

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Gittins, A. J. C., and Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *J. R. Stat. Soc. Ser. B*, 41, 148–177. doi: 10.1111/j.2517-6161.1979.tb01068.x

Hochreiter, S., Younger, A. S., and Conwell, P. R. (2001). "Learning to learn using gradient descent," in *ICANN*, volume 2130 of *Lecture Notes in Computer Science* (Springer), 87–94.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). "An efficient approach for assessing hyperparameter importance," in *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, E. P. Xing and T. Jebara (Bejing: PMLR), 754–762.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T.,et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science* 220, 671–80. doi: 10.1126/science.220.4598.671

Markram, H., Meier, K., Lippert, T., Grillner, S., Frackowiak, R., Dehaene, S., et al. (2011). Introducing the human brain project. *Proc. Comput. Sci.* 7, 39–42. doi: 10.1016/j.procs.2011.12.015

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Nawrocki, R. A., Voyles, R. M., and Shaheen, S. E. (2016). A mini review of neuromorphic architectures and implementations. *IEEE Trans. Electron Devices* 63, 3819–3829. doi: 10.1109/TED.2016.2598413

Pantazi, A., Woźniak, S., Tuma, T., and Eleftheriou, E. (2016). All-memristive neuromorphic computing with level-tuned neurons. *Nanotechnology* 27:355205. doi: 10.1088/0957-4484/27/35/355205

Rechenberg, I. (1973). *Evolutionsstrategie : Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Robert Canini, K., Shashkov, M. M., Griffiths, T. L. (2010). "Modeling transfer learning in human categorization with the hierarchical dirichlet process," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, eds J. Fürnkranz and T. Joachims (Haifa: Omnipress), 151–158. Available online at: https://icml.cc/Conferences/2010/papers/180.pdf

Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *Eur. J. Oper. Res.* 99, 89–112.

Salimans, T., Ho, J., Chen, X., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *CoRR* abs:1703.03864

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (IEEE), 1947–1950.

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *CoRR* abs:1705.06963.

Subramoney, A., Rao, A., Scherr, F., Bohnstingl, T., Jordan, J., Kopp, N., et al. (2019). IGITUGraz/L2L: v0.4.3.

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning : An Introduction*. MIT Press.

Taylor, M. E., and Stone, P. (2009). Transfer learning for reinforcement learning domains: a survey. *J. Mach. Learn. Res.* 10, 1633–1685. doi: 10.1007/978-3-642-01882-4

Wang, D., and Zheng, T. F. (2015). "Transfer learning for speech and language processing," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)* (Hong Kong: IEEE), 1225–1237. doi: 10.1109/APSIPA.2015.7415532

Wang, J. X., Kurth-Nelson, Z., Soyer, H., Leibo, J. Z., Tirumala, D., Munos, R., et al. (2016). "Learning to reinforcement learn," in *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*, eds. G. Gunzelmann, A. Howes, T. Tenbrink, and E. J. Davelaar (London: cognitivesciencesociety.org). Available online at: https://mindmodeling.org/cogsci2017/papers/0252/index.html

Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Mach. Learn.* 8, 279–292. doi: 10.1023/A:1022676722315

Wunderlich, T., Kungl, A. F., Müller, E., Hartel, A., Stradmann, Y., Aamir, S. A., et al. (2018). Demonstrating advantages of neuromorphic computation: a pilot study. *arxiv:1811.03618*. doi: 10.3389/fnins.2019.00260

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

First Error-Based Supervised Learning Algorithm for Spiking Neural Networks

Xiaoling Luo, Hong Qu, Yun Zhang and Yi Chen*

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Neural circuits respond to multiple sensory stimuli by firing precisely timed spikes. Inspired by this phenomenon, the spike timing-based spiking neural networks (SNNs) are proposed to process and memorize the spatiotemporal spike patterns. However, the response speed and accuracy of the existing learning algorithms of SNNs are still lacking compared to the human brain. To further improve the performance of learning precisely timed spikes, we propose a new weight updating mechanism which always adjusts the synaptic weights at the first wrong output spike time. The proposed learning algorithm can accurately adjust the synaptic weights that contribute to the membrane potential of desired and non-desired firing time. Experimental results demonstrate that the proposed algorithm shows higher accuracy, better robustness, and less computational resources compared with the remote supervised method (ReSuMe) and the spike pattern association neuron (SPAN), which are classic sequence learning algorithms. In addition, the SNN-based computational model equipped with the proposed learning method achieves better recognition results in speech recognition task compared with other bio-inspired baseline systems.

Keywords: spike neural networks, supervised learning, synaptic plasticity, first error learning, speech recognition

OPEN ACCESS

Edited by:
Yansong Chua,
*Institute for Infocomm Research (A*STAR), Singapore*

Reviewed by:
Angel Jimenez-Fernandez,
University of Seville, Spain
Melika Payvand,
Institute of Neuroinformatics, ETH Zurich, Switzerland

****Correspondence:***
Hong Qu
hongqu@uestc.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *26 February 2019*
Accepted: *15 May 2019*
Published: *06 June 2019*

Citation:
Luo X, Qu H, Zhang Y and Chen Y (2019) First Error-Based Supervised Learning Algorithm for Spiking Neural Networks. Front. Neurosci. 13:559. doi: 10.3389/fnins.2019.00559

1. INTRODUCTION

For years, researchers have been exploring and trying to simulate the brain's powerful and high-speed information processing capabilities and learning mechanisms. While the traditional artificial neural networks (ANNs) have achieved outstanding performance in various application fields, they assume that sensory information is represented and transmitted via the firing rate of the neuron. Nevertheless, the rate-based coding does not seem to transmit all the information associated with the rapid processing sensory tasks, such as vision, smell, and hearing stimulus modalities (Hopfield, 1995; Gautrais and Thorpe, 1998; Cariani, 2004; Mohemmed et al., 2013). A new type of artificial neural network that is dedicated to the study of more biologically plausible neuronal models and neural networks has emerged and has been well used (Wu et al., 2018a,b), which is called spiking neural networks (SNNs). On the other hand, many recent studies have shown that spike-timing neural activities exist in several areas of the brain, such as the visual cortex (Bair and Koch, 1996), the retina (Meister, 1998; Uzzell and Chichilnisky, 2004; Gollisch and Meister, 2008), and the lateral and geniculate nucleus (Reinagel and Reid, 2000). Temporally encoded SNNs that represent information as precisely timed spikes rather than mean firing rates have also been studied extensively (Maass, 1997; Andrew, 2002; Ghosh-Dastidar and Adeli, 2009b; Nguyen et al., 2012; Wang et al., 2012). Though the powerful computing performance of SNNs has been demonstrated (Keller and Hahnloser, 2009), its practical application is still limited by its computational complexity, and the learning algorithms applicable to SNNs are also generally short of high efficiency and stability. Therefore, it is of great significance to develop new effective and robust learning algorithms to take full advantage of the powerful computing performance of SNNs.

In many cases, learning behavior is thought to be performed by utilizing the error signals, i.e., the mismatches between expected and actual spiking behaviors (Thach, 1996; Bastos et al., 2012; Keller et al., 2012; Wu et al., 2019). Supervised learning based on error signals has obtained the most documented evidence in the study of the cerebellum and cerebellar cortex of the central nervous system, although the exact mechanism still remains unclear (Ito, 2000). The aim of supervised learning is to minimize the gap between actual output and expected output, and according to the different ways of reducing the gap, the existing learning algorithms of SNNs can be divided into two categories. One is to utilize rigorous mathematical analysis to derive formulas of loss reduction, and the other is to make weight updating according to the inspiration of biological mechanisms, such as the Widrow-Hoff rule (Widrow and Lehr, 1990) and the spike-timing dependent plasticity (STDP) rule (Masquelier et al., 2009), where the synaptic strength is enhanced when the presynaptic neuron elicits spikes before the postsynaptic neuron and vice versa.

Many methods based on mathematical analysis adopt the idea of gradient descent, but they define the cost function in different ways. SpikeProp (Bohte et al., 2002) minimizes the loss defined by the distance between the true firing time and the single desired firing time using gradient descent rule, and later this algorithm was improved to emit multiple spikes (Ghosh-Dastidar and Adeli, 2009a; Xu et al., 2013a). In addition to these methods, Tempotron (Gütig and Sompolinsky, 2006), an algorithm that has been proved to be effective for binary temporal classification but unable to handle the firing of multiple spikes, and some other algorithms (Zhang et al., 2018, 2019a) define the cost function as the distance between the membrane voltage and the firing threshold. Recently, there is another thought of defining cost function of multi-spike sequences. For example, Multi-Spike Tempotron (MST) (Gütig, 2016) is designed to decrease the difference between a hypothetical threshold and the fixed threshold. MST also employs the gradient descent strategy, and in each iteration the difference between the fixed biological firing threshold and the hypothetical threshold under which neurons emit the expected amount of spikes is calculated. However, it requires multiple recursive calculations to derive the hypothetical threshold, making the learning process indirect and computationally time-consuming. TDP1 and TDP2 (Yu et al., 2018) simplify the calculation of MST to some extent, which improves the learning efficiency, but there is still the problem of seeking the hypothesis threshold through iteration.

The Remote Supervised Method (ReSuMe) (Ponulak and Kasiński, 2010) is a classic algorithm that combines the STDP and anti-STDP learning rules to modulate the synaptic weights. There are also some improved algorithms to further strengthen the learning property of the ReSuMe by integrating it with delay learning (Taherkhani et al., 2015a,b, 2018), and particle swarm optimization (PSO) algorithm (Xie et al., 2014), etc. In addition, the Spike Pattern Association Neuron (SPAN) (Mohemmed et al., 2012), Chronotron E-learning (Florian, 2012), and the Precise-Spike-Driven (PSD) (Yu et al., 2013) algorithm are in a similar vein, whereby they transform spike trains or sequences into analog signals by convolution, then apply the Widrow-Hoff rule to update weights. SPAN uses a variant metric of the van Rossum metric (van Rossum, 2001) to define the distance between the actual and desired spike sequences, while Chronotron E-learning uses the Victor and Purpura metric (Victor and Purpura, 2009). SPAN transforms all the discrete input, actual and desired output spikes to continuous signals, while only input signals are convolved in PSD. Compared with algorithms requiring convolution operation, algorithms based on the perceptron rule, such as the perceptron-based spiking neuron learning rule (PBSNLR) (Xu et al., 2013b) and its improved version (Qu et al., 2015), the normalized perceptron based learning rule (NPBLR) (Xie et al., 2017), are easier to calculate. In general, these algorithms are more biologically plausible and have lower computational complexity than the algorithms based on the gradient descent rule, but they are still not very effective and robust in the task of learning target spatiotemporal spike patterns.

Except for these algorithms, the algorithm Learning Spike Sequences with Finite Precision (FP) (Memmesheimer et al., 2014) uses the existing postsynaptic potential to adjust the synaptic weights at the first unmatched time between the actual and desired output spike trains in each trial. However, the simple and crude way of weight modification makes it use less spike information and also lack good robustness in the face of noise. Then in this paper, we propose a new efficient and robust learning algorithm. The proposed algorithm not only utilizes the first wrong spike time, but also utilizes all previous spike temporal information to calculate the weight update quantities. Simulation results demonstrate that the proposed learning rule has higher learning accuracy, efficiency, and better robustness as compared with ReSuMe and SPAN. In addition, in this paper, we also put forward a dynamic decoding strategy for precise multi-spike learning algorithms. With a combination of the proposed learning algorithm and the decoding strategy, the SNN-based computational model outperforms other bio-inspired baseline systems in a speech recognition task.

The structure of the article is as follows. In section 2, after a brief introduction of the neuron model, our method is presented. In section 3, we conduct some experiments to explore the performance of the method, and the simulation results are provided. The different properties of the proposed algorithm, ReSuMe and SPAN are analyzed and compared in section 4. Finally, we draw the conclusion in section 5.

2. NEURON MODEL AND LEARNING ALGORITHM

In this section, we first introduce the spiking neuron model used in this article, then elaborate on the algorithm we proposed. Finally, the measurement used to evaluate the learning performance is introduced.

2.1. Neuron Model

Many spiking neuron models have been proposed over the years, among which conductance-based models can simulate biological neurons' dynamics accurately to a large extent but require considerable computational cost because of the inherent complexity of their expressions. By contrast, the current-based leaky integrate-and-fire (LIF) (Gerstner and Kistler, 2002) model can well simulate the dynamics of biological neurons with lower computation cost, which has made it a widely used model in many papers, including this one.

In the LIF model, learning neuron accumulates its membrane voltage $V(t)$ by integrating synaptic currents from N upstream neurons, yielding

$$V(t)=\sum_{i=1}^{N} w_i \sum_{t_i^j<t} K\left(t-t_i^j\right)-\vartheta \sum_{t_s^j<t} \exp\left(-\frac{t-t_s^j}{\tau_m}\right), \quad (1)$$

where t_i^j is the firing time of the jth spike from the ith synapse and t_s^j is the firing time of the jth spike generated by the learning neuron. ϑ is the firing threshold. w_i represents the synaptic strength of the ith synapse, and it controls the amplitude of the postsynaptic potential induced by its spike, while the kernel $K(\cdot)$ controls the shape, and it is defined as

$$K(x)=V_{norm}\left[\exp\left(-\frac{x}{\tau_m}\right)-\exp\left(-\frac{x}{\tau_s}\right)\right], \quad (2)$$

where τ_m and τ_s are the time constants of the membrane potential and the synaptic current, respectively. V_{norm} is the normalization constant that stretches the peak value of $K(\cdot)$ to unit, and it is calculated by

$$V_{norm}=\frac{\beta^{\beta/(\beta-1)}}{\beta-1}, \quad (3)$$

with $\beta=\tau_m/\tau_s$. If the voltage $V(t)$ reaches the firing threshold, it triggers a spike immediately, then this new spike causes the membrane voltage of the neuron to encounter a reset operation, which is expressed by the second term in Equation (1).

2.2. First Error Learning Algorithm

The aim of our learning algorithm is to modify the neuron's synaptic weights so that it can generate the target spike sequence corresponding to the given input spike pattern. Most existing algorithms train the neuron to fire spikes directly toward the desired times, but here we set a tolerance window with a small width ε (less than the distance between any two desired spike times) at each desired time, and by training the neuron to emit a spike within the corresponding tolerance window in chronological order, the requirement of firing target spike sequence is finally achieved. Accordingly, we present our learning method taking advantage of the idea of running synaptic modification rules only at the first wrong spike time in each trial in Memmesheimer et al. (2014).

There are different types of wrong spike times, but in general they all fall into one of the three categories and are shown in **Figure 1**:

- If there is a spike fired outside all tolerable windows, this spike time is a wrong spike time of type a;
- If there are two spikes generated within a same window, the second spike time is a wrong spike time of type b;
- If there is no spike within the desired tolerable window, the desired spike time is a wrong spike time of type c.

Following the idea of running synaptic modification rules only at the first wrong spike time in each trial, the proposed First Error Learning rule (FE-Learn) calculates weight adjustment in a new way that utilizes more temporal information between the input and output spike trains. Based on the different error types, the proposed method employs two weight updating processes. The cost function is defined as

$$E=\pm\left(\vartheta-V\left(t_{err}\right)\right), \quad (4)$$

where t_{err} is the first wrong spike time and the $\pm$ sign corresponds to weight increment and decrement, respectively.

2.2.1. Weight Increment at Desired Output Spike Times

In terms of error type c, a spike is supposed to be emitted within the tolerable window of a desired output spike time t_d^j, while it is not, so t_{err} is equal to t_d^j. Then, we apply the gradient descent method to stretch the membrane potential at time t_{err} to the threshold ϑ.

In gradient-based learning, the weight modification Δw_i is proportional to the negative of the derivative of the cost function with respect to w_i:

$$\Delta w_i=-\lambda_1\frac{dE}{dw_i}=\lambda_1\frac{dV\left(t_{err}\right)}{dw_i}, \quad (5)$$

where $\lambda_1 > 0$ is the learning rate that defines the size of the weight increment. From Equation (1), the membrane potential $V(t_{err})$ not only receives the direct influence of the synaptic weights, but also the indirect influence of them, which

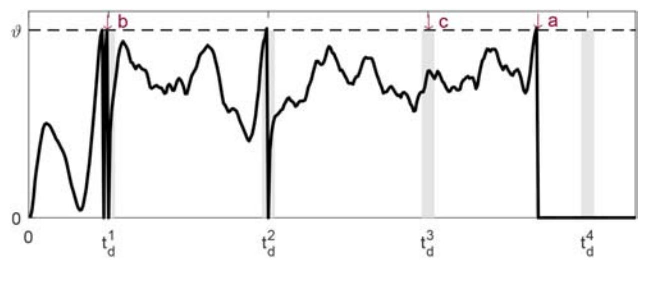


FIGURE 1 | Three error types: undesired spike outside the tolerable window **(a)**, undesired spike inside the tolerable window **(b)** and missed spike within the tolerable window **(c)**. The gray vertical bars near the desired spike times t_d^j are the respective tolerable windows.

is transmitted by the previous output spike times $t_o^j < t_{err}$, $j = 1, 2, \cdots, m$. The derivative term in Equation (5) is hence given by

$$\frac{dV(t_{err})}{dw_i} = \frac{\partial V(t_{err})}{\partial w_i} + \sum_{j=1}^{m} \frac{\partial V(t_{err})}{\partial t_o^j} \frac{dt_o^j}{dw_i}. \tag{6}$$

From Equation (1), the first term of Equation (6) can be expressed as

$$\frac{\partial V(t_{err})}{\partial w_i} = \sum_{t_i^j < t_{err}} K\left(t_{err} - t_i^j\right), \tag{7}$$

and the partial derivative in the second term is

$$\frac{\partial V(t_{err})}{\partial t_o^j} = -\frac{\vartheta}{\tau_m} \exp\left(-\frac{t_{err} - t_o^j}{\tau_m}\right), \tag{8}$$

while for the derivative dt_o^j/dw_i, applying the chain rule, we can get

$$\begin{aligned} \frac{dt_o^j}{dw_i} &= \frac{\partial t_o^j}{\partial V(t_o^j)} \frac{dV(t_o^j)}{dw_i} \\ &= \frac{\partial t_o^j}{\partial V(t_o^j)} \left(\frac{\partial V(t_o^j)}{\partial w_i} + \sum_{k=1}^{j-1} \frac{\partial V(t_o^j)}{\partial t_o^k} \frac{dt_o^k}{dw_i} \right) \\ &\approx \frac{\partial t_o^j}{\partial V(t_o^j)} \frac{\partial V(t_o^j)}{\partial w_i}, \end{aligned} \tag{9}$$

in order to save the computation cost, we eliminate the iterative computation term in Equation (9). Following the linear assumption of threshold crossing in Bohte et al. (2002), Ghosh-Dastidar and Adeli (2009a), and Yu et al. (2018), the neuron's membrane potential is thought to increase linearly in the infinitesimal time step before the firing time. Hence, there is

$$\frac{\partial t_o^j}{\partial V(t_o^j)} = -\left(\frac{\partial V(t_o^j)}{\partial t_o^j}\right)^{-1}, \tag{10}$$

where

$$\begin{aligned} \frac{\partial V(t_o^j)}{\partial t_o^j} &= \frac{\partial V(t)}{\partial t}\Big|_{t=t_o^{j-}} \\ &= \frac{V_{norm}}{\tau_s} \sum_{i=1}^{N} w_i \sum_{t_i^j < t_o^j} \exp\left(-\frac{t_o^j - t_i^j}{\tau_s}\right) \\ &- \frac{V_{norm}}{\tau_m} \sum_{i=1}^{N} w_i \sum_{t_i^j < t_o^j} \exp\left(-\frac{t_o^j - t_i^j}{\tau_m}\right) \\ &+ \frac{\vartheta}{\tau_m} \sum_{k=1}^{j-1} \exp\left(-\frac{t_o^j - t_o^k}{\tau_m}\right), \end{aligned} \tag{11}$$

and $\partial V(t_o^j)/\partial w_i$ can be solved by Equation (7), and $\partial t_o^j/\partial V(t_o^k)$ with $t_o^k < t_o^j$ can be solved by Equation (8).

Note that each actual output spike time t_o^j before the t_{err} is within the tolerable window of the corresponding desired spike time t_d^j, and there is usually a slight deviation between t_o^j and t_d^j. So the weight modification strategy based on Equation (6) may exacerbate this deviation after multiple updates, resulting in more unnecessary adjustments. In order to address this, in the actual weight adjustment, we substitute t_o^j for t_d^j in Equation (6) through Equation (11) and give a scaling factor S_r (> 0) to the second term of Equation (6) to control the weight updating at t_d^j ($< t_{err}$) not excessively (the detailed analysis is presented in section 4), which is proven to be meaningful and vital by experiments.

2.2.2. Weight Decrement at Undesired Output Spike Times

When there is a spike fired outside the tolerable window (error type a) or there is more than one spike fired inside the same tolerable window (error type b), the contributory synaptic weights should be weakened to prevent the extra spike. Instead of utilizing all the past firing spikes (actual or desired) like the case of weight increment, for error types a and b, synaptic weight decrement depends only on the error time t_{err}, i.e., the scaling rate S_r is set to zero. As a result, the second term in Equation (6) is removed, and the updating rule at undesired output spikes is defined as

$$\Delta w_i = -\lambda_2 \frac{dE}{dw_i} = -\lambda_2 \frac{dV(t_{err})}{dw_i} \approx -\lambda_2 \frac{\partial V(t_{err})}{\partial w_i}, \tag{12}$$

where $\lambda_2 > 0$ is the learning rate which defines the size of the weight decrement. $\partial V(t_{err})/\partial w_i$ is solved by Equation (7).

The intention of removing the second term in Equation (6) is to avoid disturbing the properly emitted output spikes before t_{err}. How this affects the previously emitted spikes is explained in section 4. To better illustrate the process of the proposed FE-Learn algorithm, we give a flowchart in **Figure 2**.

2.3. Metric of Learning Performance

The correlation-based metric C defined in Schreiber et al. (2003) is adopted in the next experiments to evaluate the learning performance of the learning algorithm, and it was also used in Ponulak and Kasiński (2010) and Taherkhani et al. (2015a). $C\,(0 < C < 1)$ represents the similarity degree of two vectors, and the larger the value of C, the higher the similarity between the two vectors. The metric is defined in the following equation:

$$C = \frac{\boldsymbol{v_d} \cdot \boldsymbol{v_o}}{|\boldsymbol{v_d}||\boldsymbol{v_o}|}, \tag{13}$$

where $\boldsymbol{v_o}$ and $\boldsymbol{v_d}$ are vectors which are the convolution (in discrete time) of actual and desired output spike trains by a symmetric Gaussian filter given as $f(t, \sigma) = \exp\left(-t^2/2\sigma^2\right)$, respectively. The parameter σ determining the width of the filter is set to 2 in this article. And $\boldsymbol{v_d} \cdot \boldsymbol{v_o}$ represents the dot product of the two vectors, while $|\boldsymbol{v_d}|$ and $|\boldsymbol{v_o}|$ are the Euclidean norms of them, respectively.

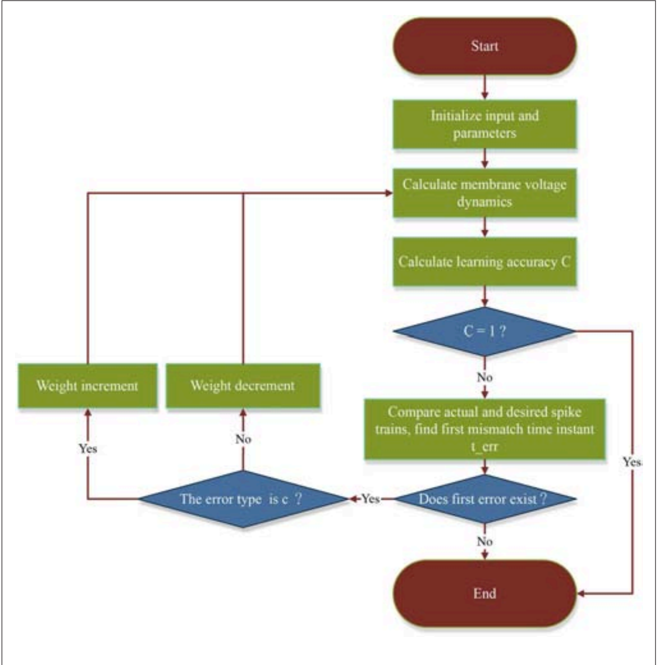

Start
Initialize input and parameters
Calculate membrane voltage dynamics
Calculate learning accuracy C
C = 1 ?
No
Compare actual and desired spike trains, find first mismatch time instant t_err
Does first error exist ?
The error type is c ?
Weight increment
Weight decrement
Yes
No
End

FIGURE 2 | The flowchart of the algorithm FE-Learn.

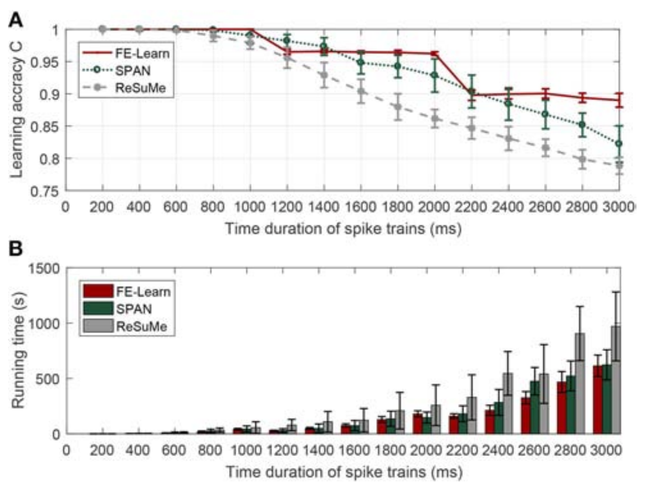


FIGURE 3 | Effect of the time duration of spike trains on learning performance. When the time duration of spike trains is in [200, 1,000], [1,200, 2,000], [2,200, 3,000] ms, the corresponding width of the tolerable window is 1, 3, and 5 ms, respectively. Learning accuracy comparison of FE-Learn, SPAN, and ReSuMe under different time duration of spike trains **(A)**, running time comparison of FE-Learn, SPAN, and ReSuMe under different time duration of spike trains **(B)**.

3. SIMULATION RESULTS

Next, we conduct extensive experiments to explore the influence of different parameters with different values on the learning performance of the FE-Learn. Moreover, the robustness in the face of noise of different intensities is tested, and finally, FE-Learn is applied to a practical speech recognition task.

3.1. Performance Evaluation of FE-Learn

The effects of several important parameters on learning performance are investigated in this section, including the time duration of spike trains, the number of synaptic inputs, and the firing rates of input and output spike trains. We compared the FE-Learn against ReSuMe and SPAN. In these simulations, the time constant of the membrane potential and the synaptic currents, τ_m and τ_s, are set to 10 and 2.5 ms, respectively. And the firing threshold and the time step are set to 1 mV and 1 ms, respectively. The synaptic weights are randomly initialized by the Gaussian distribution $N(0.01, 0.01)$. Twenty trials with different input and desired output pairs are conducted for each experiment.

3.1.1. Effect of the Time Duration

In this section, the learning neuron has 400 synaptic afferents. The aim is to train the neuron to reproduce a desired spike train with a time duration of 200 $\sim$ 3,000 ms and the length of the interval is 200 ms. Before each training trial, the desired output is a spike train with a firing rate of 100 Hz, and input spike trains with a firing rate of 10 Hz are generated according to the homogeneous Poisson processes. During each training, the maximum value of C and the running time required to reach it are recorded. After 20 training trials, the average values of all maximum C and corresponding running times are reported.

Figure 3A shows the variation trend in learning accuracies of FE-Learn, SPAN, and ReSuMe. The learning accuracies of the three algorithms can reach one when the time duration of spike trains varies from 200 to 600 ms, but when the time duration exceeds 800 ms, the learning accuracies of SPAN and ReSuMe start to decline, and the learning times increase gradually. Meanwhile, the learning accuracy of FE-Learn is limited by the width of the tolerable window ε, so it can keep constant at 1 when $\varepsilon = 1$, $C \approx 0.96$ when $\varepsilon = 3$ and $C \approx 0.89$ when $\varepsilon = 5$, and the learning accuracy drops significantly when the width of the tolerable window changes. Under the same width of the tolerable window, the learning time increases with the increase of spike train length. The general trend is that FE-Learn can obtain higher learning accuracy than SPAN and ReSuMe with less time.

3.1.2. Effect of the Number of the Synaptic Inputs

The effect of the number of the synaptic inputs is investigated in this section, and it varies from 100 to 500 with an interval of 50. The time duration of the spike trains is set to 800 ms. The desired output spike train with a firing rate of 100 Hz and input spike train with a firing rate of 10 Hz are generated according to the homogeneous Poisson processes at the beginning of each training trial.

Figure 4 shows the experimental results. As shown in **Figure 4A**, a small number of synaptic inputs lead to a low learning accuracy for both SPAN and ReSuMe—for instance, the learning accuracy of SPAN is only 0.81 and for ReSuMe it is 0.79—when the neuron is trained with only 100 synaptic inputs, but SPAN takes a very short time, and although FE-Learn with $\varepsilon = 5$ takes more time, it can achieve higher accuracy. When

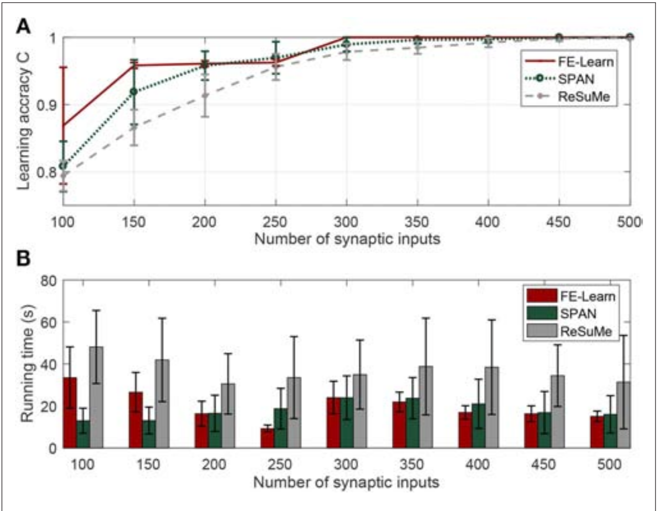


FIGURE 4 | Effect of the number of the synaptic inputs on learning performance. When the synaptic input is in 100, [150, 250], [300, 500], the corresponding width of the tolerable window is 5, 3, and 1, respectively. Learning accuracy comparison of FE-Learn, SPAN, and ReSuMe with different number of the synaptic inputs **(A)**, running time comparison of FE-Learn, SPAN, and ReSuMe with different number of the synaptic inputs **(B)**.

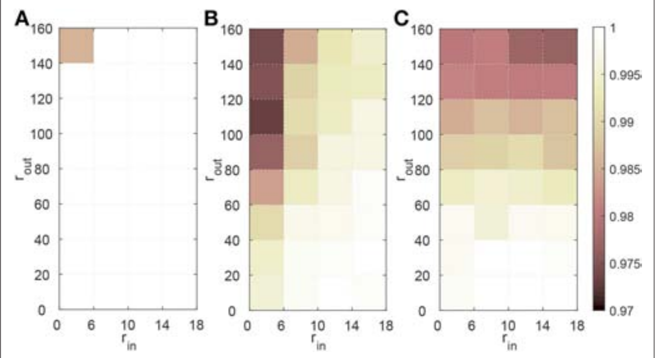


FIGURE 5 | Effect of the firing rate of the spike trains on learning performance of FE-Learn **(A)**, SPAN **(B)**, and ReSuMe **(C)**. All parameters except the firing rates of input spike trains r_{in} and the desired output spike trains r_{out} are fixed. The width of the tolerable window ε is set to 1.

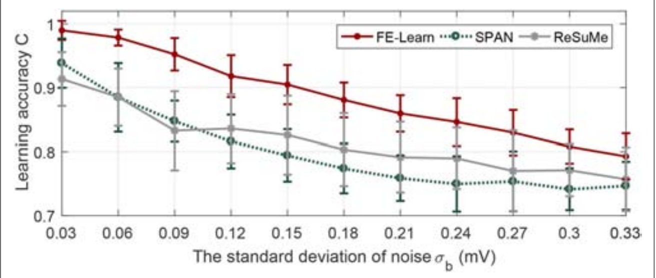


FIGURE 6 | Antinoise capability of FE-Learn, SPAN, and ReSuMe against background voltage noise. The width of the tolerable window ε is set to 1.

the number of synaptic inputs is greater than or equal to 300, the width of the tolerable window of FE-Learn is set to 1 ms. Then, the learning accuracy of it can reach 1, while the learning accuracies of SPAN and ReSuMe slowly increase to 1 with the increase of the number of synaptic inputs. Additionally, under the same width of the tolerable window, the learning time of FE-Learn can decrease with the increase of the number of the synaptic inputs. In short, FE-Learn performs better than ReSuMe both in terms of accuracy and running time, and obtains higher accuracy than SPAN with comparable time.

3.1.3. Effect of the Firing Rate

The effect of the firing rate of the spike trains is evaluated in the following experiments. For the input spike trains, the firing rates (r_{in}) are varied from 6 to 18 Hz with an interval of 4 Hz, while for the desired output spike trains the firing rates (r_{out}) vary from 20 to 160 Hz with an interval of 20 Hz. The time duration of the spike trains is 800 ms and the amount of the synaptic inputs is set to 400. In each trial, the learning continues until the algorithm converges and the averages of the maximum obtained C from 20 trials are reported in **Figure 5**.

From **Figure 5A**, the learning accuracy of FE-Learn can achieve 1 except when the firing rates of the input spike train and the desired output spike trains are 6 and 160 Hz, respectively, but even in this worst case, the accuracy still reaches 0.986. However, the performances of SPAN and ReSuMe become worse with the decrease of r_{in} and the increase of r_{out}, and their lowest accuracies are about 0.97, as shown in **Figure 5B**.

3.2. Robustness to Noise

In this section, the robustness of the neuron trained by FE-Learn and ReSuMe is investigated. The neuron has 400 synaptic inputs. The time duration of the input and expected spike trains is set as 500 ms, both of which are Poisson spike trains, and the firing rates of them are 10 and 100 Hz, respectively. After deterministic training, the response reliability of the neuron is considered in the case of adding background noise on the membrane potential and adding jittering noise on the input pattern.

3.2.1. Robustness to Background Noise on the Membrane Potential

After training, the membrane potential of the trained neuron is affected by background Gaussian white noise with mean 0 and variance $\sigma_b \in [0.03, 0.33]$ mV in this case. The variance interval is 0.03 mV, and for every value of σ_b, 20 independent experiments are conducted. The metric C is still used to measure the similarity of the actual output and desired output.

As shown in **Figure 6**, the learning accuracies of the three algorithms decrease with the increase of noise intensity. However, the correlation metric C achieved by the neuron trained by FE-Learn is consistently higher than that of SPAN and ReSuMe, confirming that the neuron trained by FE-Learn is more robust when encountering background noise.

3.2.2. Robustness to Jittering Noise on the Input Pattern

In this case, a Gaussian jitter with mean 0 and variance $\sigma_j \in [0.2, 2]$ ms is added to each input spike after deterministic training. In addition, every spike of the noisy input pattern may be randomly deleted with a probability of 0.05 while some new spikes may be randomly added into the noisy input pattern, which are generated by a 1 Hz homogeneous Poisson process. Just as before, the correlation measure C of the distance between the actual and the desired output spike sequences is calculated.

As can be seen from **Figure 7**, with the increase of the noise intensity, the correlation between the actual and the desired output spike trains shows a gradual downward trend, but for FE-Learn, it stays about 0.05 and 0.1 higher than that of SPAN and ReSuMe, respectively. Unlike before, SPAN performs better than ReSuMe when exposed to jitter noise. However, neurons trained by the FE-Learn have better anti-noise performance against jitter noise than either of them.

3.3. Effect of Learning Parameters

The width of the tolerance window ε and the scaling rate S_r are two important parameters of FE-Learn. We conduct experiments to explore the influence of them on learning efficiency and robustness of FE-Learn. Then we give a spatiotemporal spike pattern recognition experiment, and show the effect of ε on the testing performance.

3.3.1. Effect on Efficiency

In this section, the learning neuron has 400 synaptic afferents, and the time duration is 800 ms. Input pattern and target pattern are generated as in the previous experiments with a firing rate of 10 and 400, respectively. The scaling rate varies from 0 to 2 with an interval of 0.2, and the width of the tolerance window has four different values, 1, 3, 5, and 7 (under the condition that time step equals one, width equal to 2 is actually the same as width equal to 1, so there is no need to explore the situation of 2, 4, and 6). For each pair of ε and S_r, the learning continues until the algorithm converges and the average of the maximum obtained C from 20 trials are reported in **Figure 8**.

Tolerance window width determines the learning accuracy of convergence, and **Figure 8A** shows this obviously, and it also shows that no matter what the scaling rate is, the algorithm will eventually converge to the accuracy limited by the corresponding window width. From **Figures 8B,C**, we can see that, only when the tolerance window width is 1, the time of convergence increases as the scaling rate increases, and is always much higher than other cases, i.e., when the width is greater than 1, the scaling rate has little impact on the convergence speed, and the convergence time is always very small.

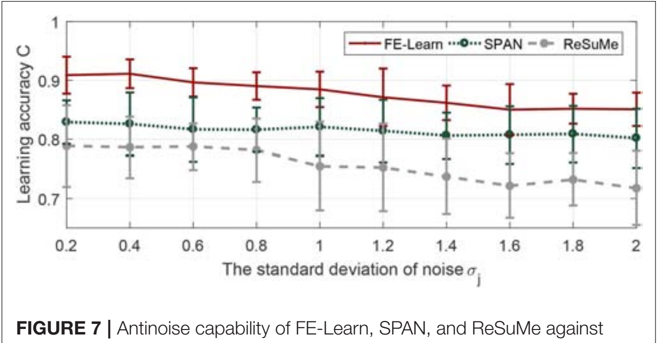


FIGURE 7 | Antinoise capability of FE-Learn, SPAN, and ReSuMe against jittering noise. The width of the tolerable window ε is set to 1.

3.3.2. Effect on Robustness

The experiment settings are the same as last section, except that the time duration is changed to 500 ms. We add background noise and jittering noise to the network after each training trial.

As seen in **Figure 9**, whether for background noise or jittering noise, the smaller the tolerance window width, the stronger the noise resistance. From **Figure 9A**, the antinoise capability against background noise becomes stronger with the increase of scaling rate, but from **Figure 9B**, the antinoise capability against jittering noise does not change obviously with the change of scaling rate.

Combined with **Figures 8**, **9**, when the window width is greater than 1, FE-Learn can converge rapidly and the convergence speed is not sensitive to the scaling rate, but increasing it can improve the antinoise performance to background noise. When the width is 1, the convergence speed of the algorithm is very slow, and the smaller the scaling rate is, the faster the convergence speed is, but the worse the antinoise performance to background noise is.

3.3.3. Effect of the Width of Tolerance Window on Overfitting

In this section, we conduct experiments to investigate the effect of the width of tolerance window on overfitting. Three different

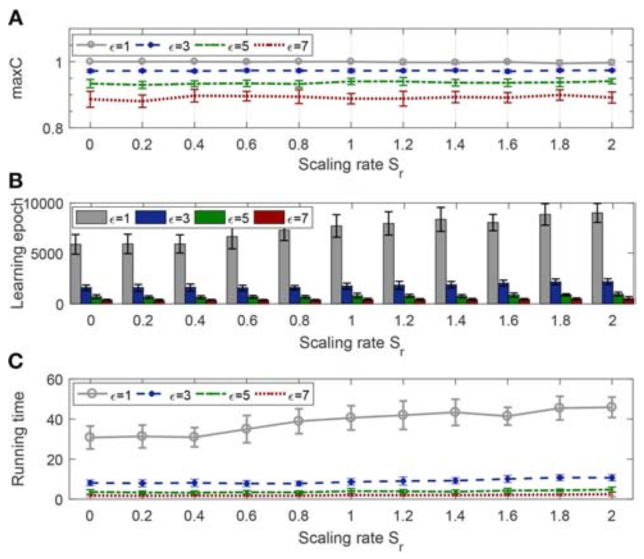


FIGURE 8 | Effect of tolerance window width and scaling rate on learning efficiency. The evaluation index includes learning accuracy **(A)**, the number of epochs **(B)**, and the running time **(C)**.

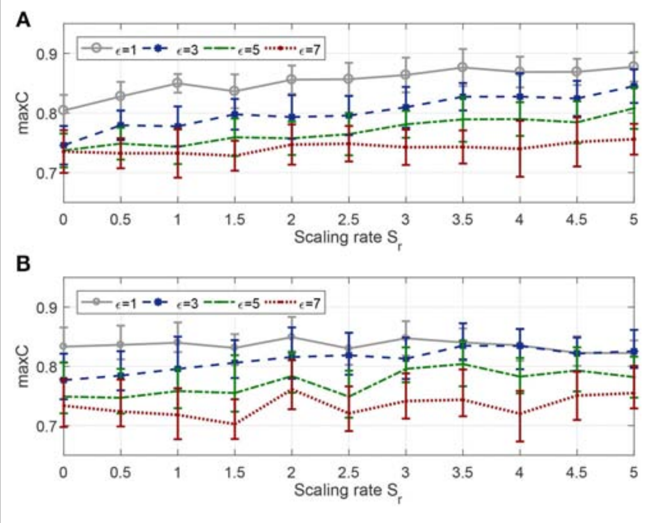


FIGURE 9 | Effect of tolerance window width and scaling rate on robustness. **(A)** Antinoise capability against background noise with standard deviation $\sigma_b = 0.2$. **(B)** Antinoise capability against jittering noise with standard deviation $\sigma_j = 1$.

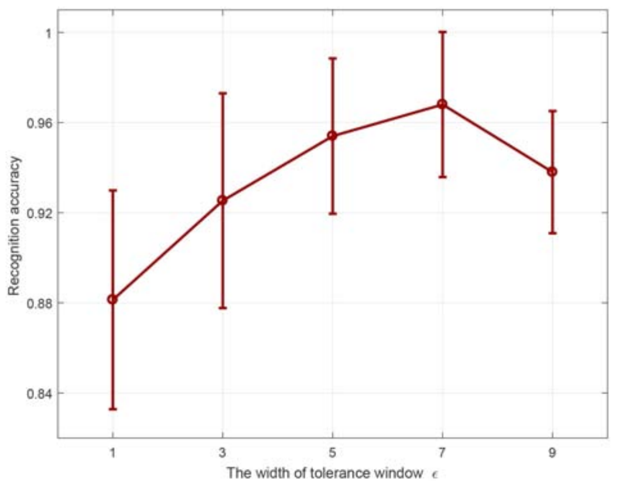


FIGURE 10 | Effect of tolerance window width on overfitting.

spatiotemporal spike patterns are randomly generated with 400 synaptic afferents, all of which are triggered at 5 Hz. The time duration of each spatiotemporal spike pattern is 200 ms. For each spike pattern, 25 samples are generated for training by adding a jitter noise drawn from a Gaussian distribution with a standard deviation of 3 ms, resulting in a training set with 3 × 25 samples. The test set is obtained in the same way. The learning neuron is trained to emit the corresponding desired output spike trains ([5:15:170], [15:15:180], [25:15:190]) in response to the three kinds of spike patterns. When the actual output spike train is most similar to the desired output spike train of a category, then the input pattern is classified into that category. For each ε, the average recognition accuracy on the test set from 20 trials is reported in **Figure 10**.

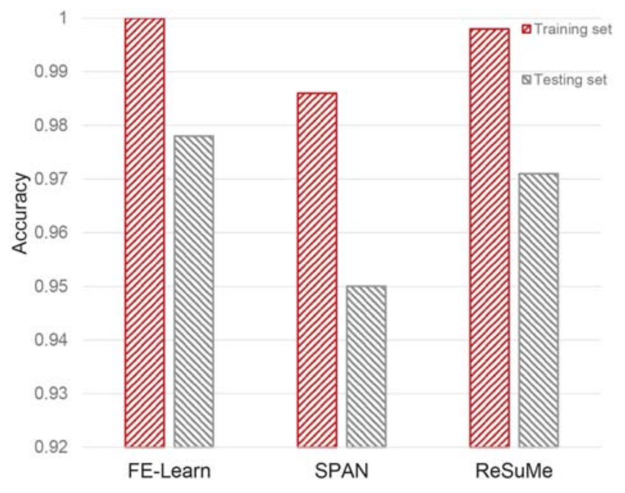


FIGURE 11 | Classification capability of FE-Learn, SPAN, and ReSuMe on spatiotemporal spike patterns.

As shown in **Figure 10**, when ε is less than or equal to 7 ms, the classification accuracy on the test set increases with the increase of window width. This is because a smaller window means more rigorous learning on the training set, which will lead to overfitting and reduce the generalization on the test set. For example, when the window width is 7 ms, the mean recognition accuracy on the test set is 96%. However, when the window width is 1 ms, the accuracy is only about 88%. On the other hand, an overly large window will make the training insufficient, thus reducing the recognition accuracy. For instance, the recognition accuracy decreases to 93.80% when the window width is 9 ms. In a nutshell, a relatively large ε generalizes better, and the recognition accuracy on the unseen data is higher.

3.4. Classification Task

3.4.1. Spatiotemporal Spike Pattern Classification

In this experiment, we investigate the ability of the proposed FE-Learn in classifying spatiotemporal patterns. The setup for the experiment is the same as in section 3.3.3. The aim of the task is to classify three different spatiotemporal spike patterns. Both the training set and test set contain 3 × 25 samples. For each algorithm, after 300 learning epochs on the training set, the classification performance on the training set and test set is tested. The results are shown in **Figure 11**.

As can be seen from **Figure 11**, the classification accuracies of FE-learn, SPAN, and ReSuMe on the training set are 1, 0.986, and 0.998 while those on the test set are 0.978, 0.95, and 0.971, respectively. FE-Learn achieves better performance in both the training set and test set. On the other hand, from the respective differences between the training accuracy and the testing accuracy (0.022 for FE-Learn, 0.036 for SPAN, 0.027 for ReSuMe), FE-Learn has a better generalization ability.

3.4.2. Speech Classification

SNNs have great advantages in handling temporally rich signals since they can transform the spatiotemporal information into desired output spike patterns, which means that SNNs are well-suited for realistic tasks such as motion and speech recognition. In order to verify the capability of FE-Learn, the spiking neurons trained by the algorithm are used to conduct a spoken digit classification task. In this work, we investigate the TIDIGITS corpus (Leonard and Doddington, 1993), one of the most commonly used data sets in benchmarking speech recognition algorithms. The utterances of this data set were collected from speakers who come from 22 different dialectical regions and are digit sequences, containing 11 words: "zero," "one," · · · , "nine," and "oh."

In this case, the threshold encoding mechanism (Gütig et al., 2009) is adopted to encode the speech data into spike patterns, and the encoding mode is the same as that in Zhang et al. (2019b). Firstly, a Constant-Q Transform (CQT) cochlear filter bank (Pan et al., 2018) is used to filter the original speech waveform to get a spectrogram. Then, the spectrogram is divided into multiple frequency bins. For each bin, a cochlear filter of the corresponding frequency is used to filter it into a series of spikes by recording events that cross thresholds up and down. Finally, the spikes filtered by all cochlear filters are vertically integrated to obtain a complete input spike pattern. Referring to the visualization processing tool of auditory information provided in Dominguez-Morales et al. (2016), a visual representation of this process is given in **Figure 12**. In the experiment, the training set and test set include 2,464 and 2,486 speech spike patterns, respectively.

The computational model used here is shown in **Figure 13**. There are eleven groups of output neuron in the classification layer, and each group contains ten neurons, which correspond to the same category. The goal of this experiment is to train the target group of neurons to emit a desired spike train when receiving the input patterns of the corresponding category, and to remain silent otherwise. However, it is not clear how to determine the target output spike train corresponding to each category as each speech digit category contains many different sub-patterns and the differences between these sub-patterns make a fixed desired output spike train impractical. To resolve this problem, a strategy for dynamically determining the target spike train is proposed as follows.

When entering a training input pattern, we record the membrane voltage traces of target neurons and non-target neurons. The desired spike trains T_d and the first wrong time t_{err} are defined as follows.

1. For the non-target neurons: $T_d = \varnothing$.
   - If no spike is generated, no learning is required.
   - If the actual output spike trains $T_o \neq \varnothing$, then the first wrong spike time t_{err} is the first actual output spike time.
2. For the non-target neurons: T_d is dynamically determined, and suppose t_{max} is the time instant when the maximum membrane voltage V_{max} under the threshold is reached. ϑ_e ($< \vartheta$) is a pre-defined encoding threshold.
   - If no spike is generated, $T_d = \{t_{max}\}$, then obviously, $t_{err} = t_{max}$.
   - If the actual output spike trains $T_o \neq \varnothing$ and V_{max} is above the pre-defined encoding threshold ϑ_e, then $T_d = T_o \cup \{t_{max}\}$, $t_{err} = t_{max}$.

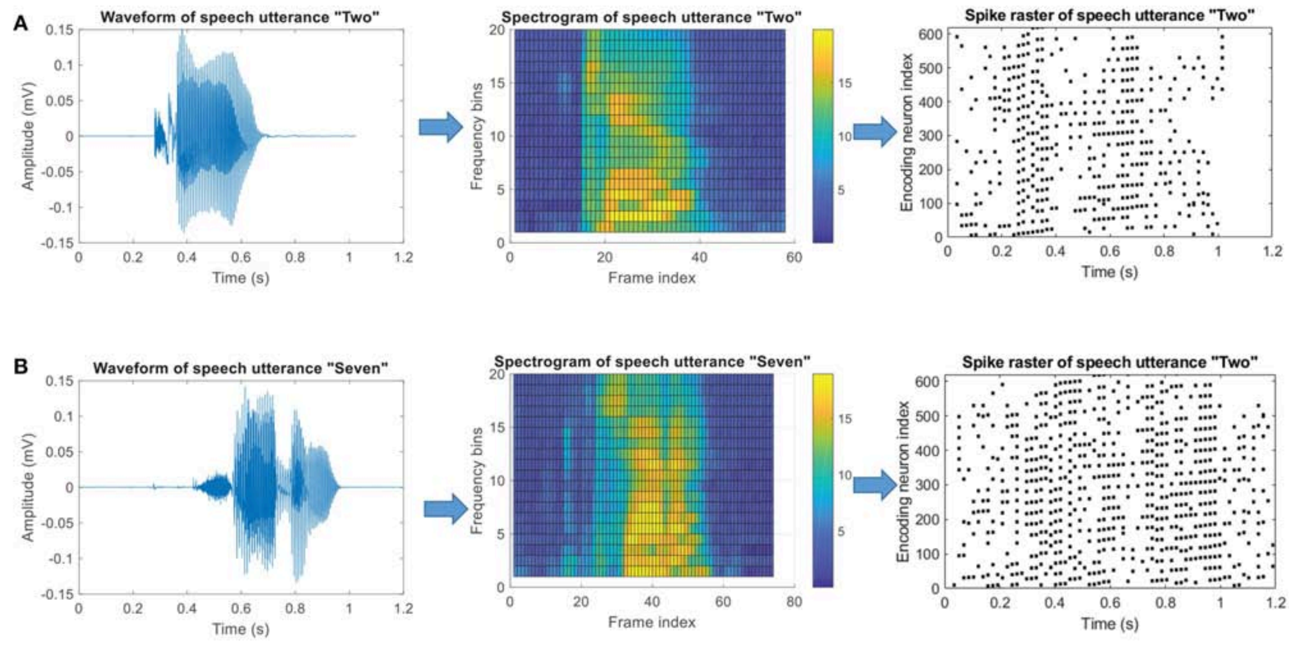


FIGURE 12 | Threshold coding mechanism of speech data. **(A)** The Encoding process of a speech utterance "Two." **(B)** The Encoding process of a speech utterance "Seven." The left column shows the original speech waveforms, the middle column shows corresponding spectrograms and the right column shows the final encoded spike pattern.

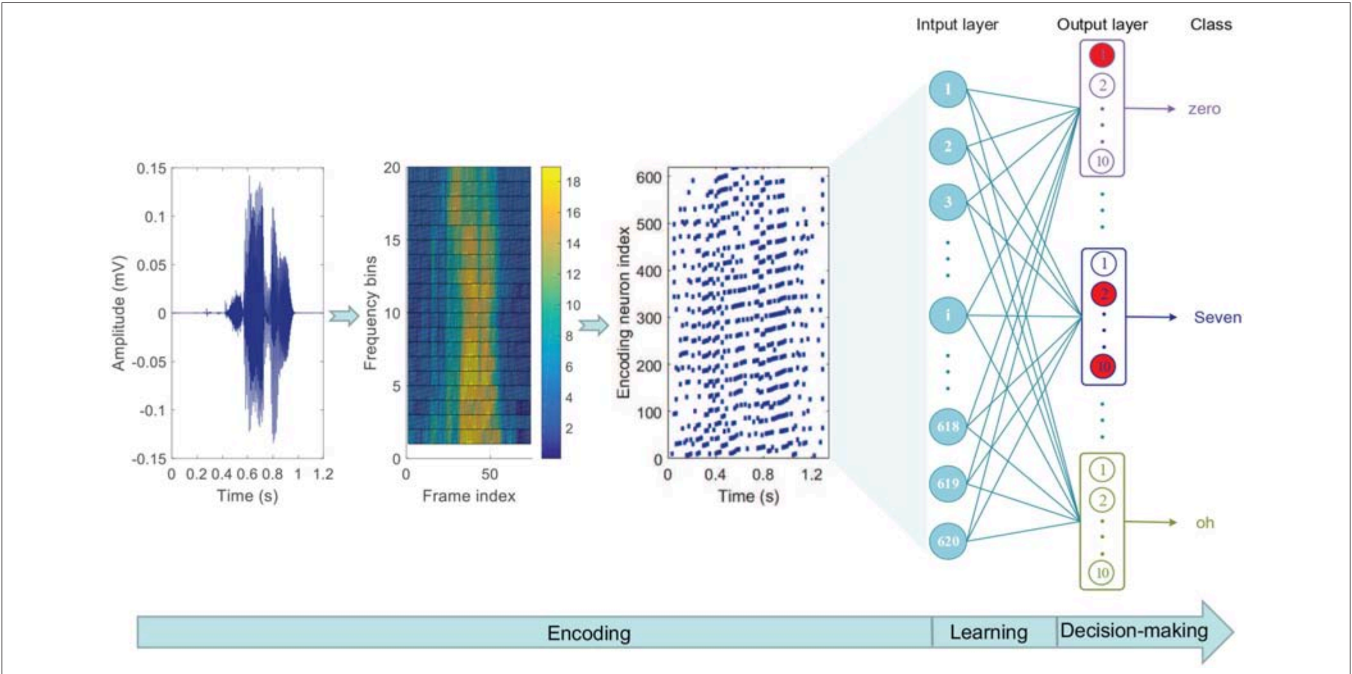


FIGURE 13 | Network architecture of speech classification. The three diagrams on the left show the encoding process of the speech data and the network structure is on the right. The input layer contains 620 neurons and the output layer contains 11 groups of neuron (mullion), corresponding to 11 output categories, and each group is composed of ten neurons. Among the ten output neurons with the same serial number in these 11 groups, the one that emits the most spikes is the "activated" neuron (red circle). The output category belongs to the group with the largest number of "activated" neurons.

- If the actual output spike trains $T_o \neq \varnothing$ and V_{max} is below the pre-defined encoding threshold ϑ_e, then $T_d = T_o$ and no learning is required.

According to the defined T_d and t_{err}, the corresponding weight updating formula is called for learning. During the test, the output category belongs to the group with the largest number of activated neurons (red neuron shown in the output layer in **Figure 13**). Moreover, the training strategy with margins in Gütig (2016) is applied in this work. We also test the performance of ReSuMe and SPAN on this task with the same network configuration, encoding method, and training strategy as FE-Learn.

As shown in **Table 1**, the spiking convolutional neural network (Tavanaei and Maida, 2016) and the deep recurrent network (Neil and Liu, 2016) perform well in this speech recognition task, and they can obtain an accuracy of 96 and 96.1%, respectively. However, compared with their complex network structures, the computational model we used here is very simple while the accuracy of our method is higher than others. As shown in **Table 1**, the single layer spiking neural network with the proposed FE-Learn algorithm obtains an accuracy of 96.42%, which is superior to other biologically motivated baselines, as well as ReSuMe and SPAN with the same network structure, encoding scheme, and training strategy. The excellent performance of FE-Learn shows its great potential in practical application.

TABLE 1 | Comparison of speech recognition performance among several frameworks.

Model	Accuracy
Spiking CNN and HMM (Tavanaei and Maida, 2016)	96.00%
Single-layer SNN and SVM (Tavanaei and Maida, 2017)	91.00%
AER Silicon Cochlea and Deep RNN (Neil and Liu, 2016)	96.10%
Liquid State Machine (Zhang et al., 2015)	92.30%
AER Silicon Cochlea and SVM (Abdollahi and Liu, 2011)	95.58%
Auditory Spectrogram and SVM (Abdollahi and Liu, 2011)	78.73%
Single-layer SNN with SPAN	91.22%
Single-layer SNN with ReSuMe	93.52%
Single-layer SNN with FE-Learn	96.42%

Additionally, in order to investigate the performance of FE-Learn in more complex cases, we also conduct speech classification experiments of the three algorithms with different input noise intensities. The standard deviation of jitter noise added to the input spike pattern increases from 0.5 to 5 ms with an interval of 0.5 ms. As shown in **Figure 14**, the classification accuracy of the proposed FE-Learn is 94.69% even when the noise intensity is 5 ms, which is much higher than ReSuMe and SPAN with the same noise level. Therefore, the robustness of the FE-Learn is better than ReSuMe and SPAN in practical application.

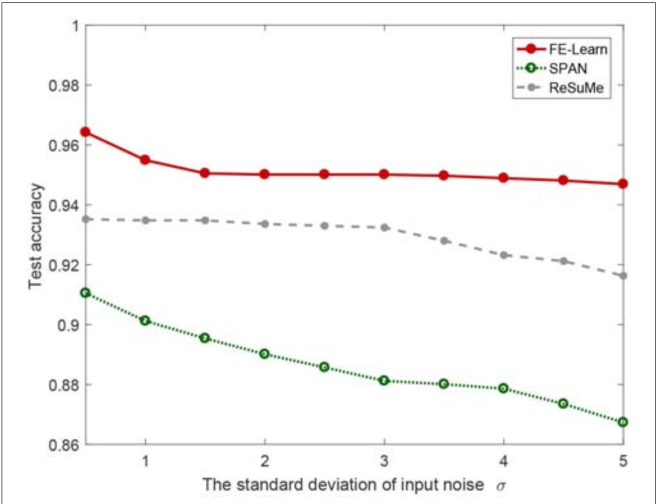


FIGURE 14 | Speech recognition performance of FE-learn, SPAN, and ReSuMe in the test set in the face of input noise.

4. DISCUSSION

In this section, we first analyze the difference between the three algorithms and explain the role of the parameter S_r through a concrete example. Then we figure out the reasons that contribute to FE-Learn's better performance over ReSuMe and SPAN in accuracy, computation time, and generalization.

The membrane potential curves before and after a single weight updating have been shown in **Figures 15A,C**, respectively. In **Figure 15B**, the synaptic learning curves depict the spike-timing dependence of weight adjustment at time t_{err}. ReSuMe has an exponential learning curve (the gray dashed line), which means that the closer the input spike time is to t_{err}, the larger the synaptic weight update is. However, due to the existence of the time constants of the membrane voltage and synaptic current, the input spike closest to t_{err} does not make the largest contribution to the membrane voltage at t_{err}, so the learning of ReSuMe does not serve the aim very well. As for SPAN, we depict its spike-timing dependence curve (green dotted line) of weight adjustment with α-kernel in Mohemmed et al. (2012) at time t_{err}. From **Figure 15A**, each actual output spike time before the t_{err} is within the tolerable window of the corresponding desired spike time. Accordingly, the convolution of the error is very small, resulting in very little weight change at t_{err}. The shape of the learning curve is determined by the convolution kernel, and the inconsistency between the convolution kernel and the current kernel of the neuron model can also lead to mismatching between the weight change of the synaptic and its potential contribution.

As we already know, FE-learn with $S_r = 1$ utilizes all the spike times before t_{err} to calculate weight increment, so the learning curve of it has multiple crests compared with that of FE-learn with $S_r = 0$ which has one crest. It means that the former would promote the synaptic weights whose input spikes happened before t_d^3 with a larger amount, but for those spikes fired between t_d^3 and t_d^4, the weight updates are the same (the red solid line and the blue dashed line coincide). As shown in **Figure 15C**, the membrane potential at t_d^4 is successfully raised in all cases, and the spike times before t_d^4 are pushed forward a little bit. But for FE-learn with $S_r = 1$, this is more obvious than others because of the greater weight updates and thus the greater voltages at these times, which means that it is more robust to noise disturbance. However, an overly strong weight update may cause the previous output spikes to be removed from the corresponding tolerable windows, so the appropriate strength of weight adjustment at previous desired spike times which is controlled by the scaling factor S_r is crucial. As for the case of weight decrement, we only want to reduce the membrane voltage at t_{err}, but do not want the previously correctly emitted spikes to be affected, so setting S_r to zero is reasonable.

As shown in the experimental results, FE-Learn achieves a higher learning accuracy with less training time and has a better generalization. First of all, the reason for the high accuracy of our method is that our method follows the BPBA (Bigger PSP, Bigger Adjustment) (Xu et al., 2013a) principle to effectively overcome learning interference among multiple desired spikes, while the weight update rules in ReSuMe and SPAN cannot be combined with the BPBA principle. Besides, to improve the efficiency of the program, we have calculated and stored the PSPs (Postsynaptic potentials) of every time step before training. For example, when the time duration is T, the time step is dt and the number of the synaptic inputs is N, storing the calculated PSPs requires $N \cdot T/dt$ storage units. For the three algorithms, the calculation of the neuron dynamics and weight adjustments are all based on the stored PSPs, and the additional memory costs required by them are very small, implying that they have a similar memory overhead. On the other hand, in each training epoch, ReSuMe makes multiple weight adjustments at each desired and actual firing time, while SPAN changes weight at each time step. However, FE-Learn only makes a weight adjustment once at t_{err} in one epoch, and the membrane potential after t_{err} does not need to be calculated in our experiments. This is the reason that FE-Learn requires less computation time. Finally, as the constraint on the tolerable window for spiking loosens, the generalization ability of proposed FE-learn learning is much better than others. This is the reason for the better results in **Figure 11**.

5. CONCLUSION

The proposed FE-Learn is designed for identifying spatiotemporal spike patterns, i.e., the neuron is trained to output the specific spike sequence for the given input spike pattern. FE-Learn adjusts the synaptic weights at the first wrong output spike time, and only when the trained neuron correctly fires the first spike at the desired time does FE-Learn begin to focus on adjusting the weights to fire the second desired spike. The adjustment of the synaptic weight is proportional

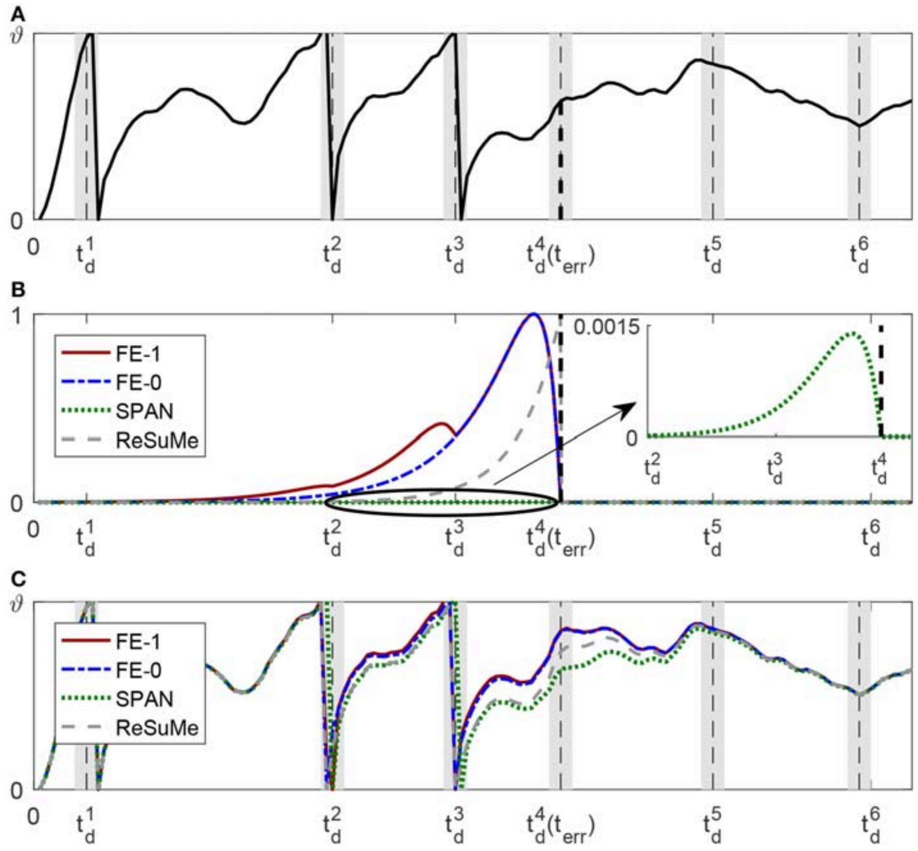


FIGURE 15 | Comparison between FF-learn, SPAN, and ReSuMe for one weight updating. The neuron has been trained to elicit the first three spikes in corresponding tolerable windows, which are represented by the gray shadow region, the desired spike times are located at the middle of the windows. The black vertical dashed lines represent the first wrong time t_{err}. **(A)** Membrane dynamics of the neuron before this learning. **(B)** Synaptic learning curves of FE-Learn with scaling rate $S_r = 1$ (red solid line) FE-Learn with scaling rate $S_r = 0$ (blue dotted line), SPAN (green dotted line), and ReSuMe (gray dashed line). **(C)** Membrane dynamics of the neuron after one learning using different rules.

to the derivative of the membrane voltage of the first wrong time with respect to the synapse. These three error types described above actually belong to two types: one is at the desired spike time, the other is at the actual spike time. They correspond to the two opposite cases of increasing and decreasing synaptic weights. For the first case, the desired spike times before the wrong spike time are also used to calculate the derivative, but for the second case, only the wrong spike time is used.

Although the proposed FE-Learn has reliable performance in the experiments, the inherent properties of this algorithm make it converge to the narrow window of the desired spike times, and it is difficult to emit a precisely timed spike. Hence we will explore how to balance the width of the window (accuracy) and the learning speed in the next work. Furthermore, extending FE-Learn to multi-layer deep spiking neural networks is another interesting future direction to explore.

DATA AVAILABILITY

The datasets analyzed for this study can be found in the TIDIGITS speech corpus https://catalog.ldc.upenn.edu/LDC93S10.

AUTHOR CONTRIBUTIONS

XL performed the experiments and writing. XL, HQ, YC, and YZ contributed to the experiment's design and interpretation of the results.

FUNDING

This work was supported in part by the National Science Foundation of China under Grant 61573081 and Grant 61806040, and in part by the Foundation for Youth Science and Technology Innovation Research Team of Sichuan Province under Grant 2016TD0018.

REFERENCES

Abdollahi, M., and Liu, S.-C. (2011). "Speaker-independent isolated digit recognition using an aer silicon cochlea[c]," in *Biomedical Circuits & Systems Conference IEEE* (La Jolla, CA).

Andrew, A. M. (2002). Spiking neuron models: single neurons, populations, plasticity. *Kybernetes* 4, 277C280. doi: 10.1108/k.2003.06732gae.003

Bair, W., and Koch, C. (1996). Temporal precision of spike trains in extrastriate cortex of the behaving macaque monkey. *Neural Comput.* 8:1185. doi: 10.1162/neco.1996.8.6.1185

Bastos, A. M., Usrey, W. M., Adams, R. A., Mangun, G. R., Fries, P., and Friston, K. J. (2012). Canonical microcircuits for predictive coding. *Neuron* 76, 695–711. doi: 10.1016/j.neuron.2012.10.038

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Cariani, P. A. (2004). Temporal codes and computations for sensory representation and scene analysis. *IEEE Trans. Neural Netw.* 15, 1100–1111. doi: 10.1109/TNN.2004.833305

Dominguez-Morales, J. P., Jimenez-Fernandez, A., Dominguez-Morales, M., and Jimenez-Moreno, G. (2016). Navis: neuromorphic auditory visualizer tool. *Neurocomputing* 237, 418–422. doi: 10.1016/j.neucom.2016.12.046

Florian, R. A. V. (2012). The chronotron: a neuron that learns to fire temporally precise spike patterns. *PLoS ONE* 7:e40233. doi: 10.1371/journal.pone.0040233

Gautrais, J., and Thorpe, S. (1998). Rate coding versus temporal order coding: a theoretical approach. *Biosystems* 48, 57–65. doi: 10.1016/S0303-2647(98)00050-1

Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models*. Cambridge: Cambridge University Press.

Ghosh-Dastidar, S., and Adeli, H. (2009a). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. *Neural Netw.* 22, 1419–1431. doi: 10.1017/CBO9780511815706

Ghosh-Dastidar, S., and Adeli, H. (2009b). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308. doi: 10.1142/S0129065709002002

Gollisch, T., and Meister, M. (2008). Rapid neural coding in the retina with relative spike latencies. *Science* 319, 1108–1111. doi: 10.1126/science.1149639

Gütig, R. (2016). Spiking neurons can discover predictive features by aggregate-label learning. *Science* 351:aab4113. doi: 10.1126/science.aab4113

Gütig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* 9, 420–428. doi: 10.1038/nn1643

Gütig, R., Sompolinsky, H., and Deweese, M. R. (2009). Time-warp-invariant neuronal processing. *PLoS Biol.* 7:e1000141. doi: 10.1371/journal.pbio.1000141

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature* 376, 33–36. doi: 10.1038/376033a0

Ito, M. (2000). Mechanisms of motor learning in the cerebellum. *Brain Res.* 886, 237–245. doi: 10.1016/S0006-8993(00)03142-5

Keller, G. B., Bonhoeffer, T., and Hübener, M. (2012). Sensorimotor mismatch signals in primary visual cortex of the behaving mouse. *Neuron* 74, 809–815. doi: 10.1016/j.neuron.2012.03.040

Keller, G. B., and Hahnloser, R. H. (2009). Neural processing of auditory feedback during vocal practice in a songbird. *Nature* 457, 187–90. doi: 10.1038/nature07467

Leonard, R. G., and Doddington, G. (1993). *Tidigits Speech Corpus*. Philadelphia, PA: Linguistic Data Consortium.

Maass, W. (1997). Network of spiking neurons: the third generation of neural network models. *Trans. Soc. Comput. Simul. Int.* 14, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive stdp-based spike pattern learning. *Neural Comput.* 21:1259. doi: 10.1162/neco.2008.06-08-804

Meister, M. (1998). "Refractoriness and neural precision," in *Conference on Advances in Neural Information Processing Systems* (Denver, CO), 110–116.

Memmesheimer, R. M., Ran, R., ölveczky, B., and Sompolinsky, H. (2014). Learning precisely timed spikes. *Neuron* 82, 925–938. doi: 10.1016/j.neuron.2014.03.026

Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. (2012). Span: spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* 22:1250012. doi: 10.1142/S0129065712500128

Mohemmed, A., Schliebs, S., Matsuda, S., and Kasabov, N. (2013). Training spiking neural networks to associate spatio-temporal inputcoutput spike patterns. *Neurocomputing* 107, 3–10. doi: 10.1016/j.neucom.2012.08.034

Neil, D., and Liu, S. C. (2016). "Effective sensor fusion with event-based sensors and deep network architectures," in *IEEE International Symposium on Circuits and Systems* (Montréal, QC).

Nguyen, V. A., Starzyk, J. A., Goh, W. B., and Jachyra, D. (2012). Neural network structure for spatio-temporal long-term memory. *IEEE Trans. Neural Netw. Learn. Syst.* 23, 971–983. doi: 10.1109/TNNLS.2012.2191419

Pan, Z., Li, H., Wu, J., and Chua, Y. (2018). "An event-based cochlear filter temporal encoding scheme for speech signals," in *2018 International Joint Conference on Neural Networks(IJCNN)* (Rio de Janeiro), 1–8.

Ponulak, F., and Kasiński, A. (2010). Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Qu, H., Xie, X., Liu, Y., Zhang, M., and Li, L. (2015). Improved perception-based spiking neuron learning rule for real-time user authentication. *Neurocomputing* 151, 310–318. doi: 10.1016/j.neucom.2014.09.034

Reinagel, P., and Reid, R. C. (2000). Temporal coding of visual information in the thalamus. *J. Neurosci.* 20:5392. doi: 10.1523/JNEUROSCI.20-14-05392.2000

Schreiber, S., Fellous, J. M., Whitmer, D., Tiesinga, P., and Sejnowski, T. J. (2003). A new correlation-based measure of spike timing reliability. *Neurocomputing* 52, 925–931. doi: 10.1016/S0925-2312(02)00838-X

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015a). Dl-resume: a delay learning-based remote supervised method for spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3137–3149. doi: 10.1109/TNNLS.2015.2404938

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015b). "Multi-dl-resume: multiple neurons delay learning remote supervised method," in *International Joint Conference on Neural Networks* (Killarney), 1–7.

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2018). A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 2018, 1–14. doi: 10.1109/TNNLS.2018.2797801

Tavanaei, A., and Maida, A. (2017). "Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals," in *International Conference on Neural Information Processing* (Guangzhou), 99–908.

Tavanaei, A., and Maida, A. S. (2016). A spiking network that learns to extract spike signatures from speech signals. *Neurocomputing* 240, 191–199. doi: 10.1016/j.neucom.2017.01.088

Thach, W. T. (1996). On the specific role of the cerebellum in motor learning and cognition: clues from pet activation and lesion studies in man. *Behav. Brain Sci.* 19, 411–433. doi: 10.1017/S0140525X00081504

Uzzell, V. J., and Chichilnisky, E. J. (2004). Precision of spike trains in primate retinal ganglion cells. *J. Neurophysiol.* 92, 780–789. doi: 10.1152/jn.01171.2003

van Rossum, M. C. (2001). A novel spike distance. *Neural Comput.* 13, 751–763. doi: 10.1162/089976601300014321

Victor, J. D., and Purpura, K. P. (2009). Metric-space analysis of spike trains: theory, algorithms and application. *Netw. Comput. Neural Syst.* 8, 127–164. doi: 10.1088/0954-898X/8/2/003

Wang, W., Subagdja, B., Tan, A. H., and Starzyk, J. A. (2012). Neural modeling of episodic memory: encoding, retrieval, and forgetting. *IEEE Trans. Neural Netw. Learn. Syst.* 23, 1574–1586. doi: 10.1109/TNNLS.2012.2208477

Widrow, B., and Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proc. IEEE* 78, 1415–1442. doi: 10.1109/5.58323

Wu, J., Chua, Y., and Li, H. (2018a). "A biologically plausible speech recognition framework based on spiking neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro: IEEE), 1–8.

Wu, J., Chua, Y., Zhang, M., Li, H., and Tan, K. C. (2018b). A spiking neural network framework for robust sound classification. *Front. Neurosci.* 12:836. doi: 10.3389/fnins.2018.00836

Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., and Li, H. (2019). Deep spiking neural network with spike count based learning rule. *arXiv [preprint]. arXiv:1902.05705*.

Xie, X., Hong, Q., Liu, G., and Liu, L. (2014). "Recognizing human actions by using the evolving remote supervised method of spiking neural networks," in *International Conference on Neural Information Processing* (Kuching).

Xie, X., Qu, H., Liu, G., and Zhang, M. (2017). Efficient training of supervised spiking neural networks via the normalized perceptron based learning rule. *Neurocomputing* 241, 152–163. doi: 10.1007/978-3-319-12637-1_46

Xu, Y., Zeng, X., Han, L., and Yang, J. (2013a). A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Netw.* 43, 99–113. doi: 10.1016/j.neunet.2013.02.003

Xu, Y., Zeng, X., and Zhong, S. (2013b). A new supervised learning algorithm for spiking neurons. *Neural Comput.* 25:1472V1511. doi: 10.1162/NECO_a_00450

Yu, Q., Li, H., and Tan, K. C. (2018). Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity. *IEEE Trans. Cybern.* 49, 2178–2189. doi: 10.1109/TCYB.2018.2821692

Yu, Q., Tang, H., Tan, K. C., and Li, H. (2013). Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *PLoS ONE* 8:e78318. doi: 10.1371/journal.pone.0078318

Zhang, M., Hong, Q., and Xie, X. (2018). Empd: an efficient membrane potential driven supervised learning algorithm for spiking neurons. *IEEE Trans. Cogn. Dev. Syst.* 10, 151–162. doi: 10.1109/TCDS.2017.2651943

Zhang, M., Qu, H., Belatreche, A., Chen, Y., and Yi, Z. (2019a). A highly effective and robust membrane potential-driven supervised learning method for spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 123–137. doi: 10.1109/TNNLS.2018.2833077

Zhang, M., Wu, J., Chua, Y., Luo, X., Pan, Z., Liu, D., et al. (2019b). "MPD-AL: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons," in *Thirty-Third AAAI Conference on Artificial Intelligence* (Honolulu, HI).

Zhang, Y., Li, P., Jin, Y., and Choe, Y. (2015). A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 2635–2649. doi: 10.1109/TNNLS.2015.2388544

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 27 June 2019
doi: 10.3389/fnins.2019.00656

A Spike Time-Dependent Online Learning Algorithm Derived From Biological Olfaction

Ayon Borthakur[1]* *and Thomas A. Cleland*[2]

[1] Computational Physiology Laboratory, Field of Computational Biology, Cornell University, Ithaca, NY, United States,
[2] Computational Physiology Laboratory, Department of Psychology, Cornell University, Ithaca, NY, United States

We have developed a spiking neural network (SNN) algorithm for signal restoration and identification based on principles extracted from the mammalian olfactory system and broadly applicable to input from arbitrary sensor arrays. For interpretability and development purposes, we here examine the properties of its initial feedforward projection. Like the full algorithm, this feedforward component is fully spike timing-based, and utilizes online learning based on local synaptic rules such as spike timing-dependent plasticity (STDP). Using an intermediate metric to assess the properties of this initial projection, the feedforward network exhibits high classification performance after few-shot learning without catastrophic forgetting, and includes a *none of the above* outcome to reflect classifier confidence. We demonstrate online learning performance using a publicly available machine olfaction dataset with challenges including relatively small training sets, variable stimulus concentrations, and 3 years of sensor drift.

Keywords: SNN, online learning, olfaction, STDP, local learning, spike time coding

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Hesham Mostafa,
University of California, San Diego,
United States
Thomas Nowotny,
University of Sussex, United Kingdom

Correspondence:
Ayon Borthakur
ab2535@cornell.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *02 February 2019*
Accepted: *07 June 2019*
Published: *27 June 2019*

Citation:
Borthakur A and Cleland TA (2019) A Spike Time-Dependent Online Learning Algorithm Derived From Biological Olfaction. Front. Neurosci. 13:656. doi: 10.3389/fnins.2019.00656

INTRODUCTION

Convolutional networks have enabled tremendous progress in image recognition. However, analogous problems in high-dimensional modalities that lack the two-dimensional internal structure of visual images are not well-addressed by these networks, and the development of brain-mimetic network-based signal identification strategies in such modalities has lagged. This is unfortunate, as there are innumerable applications for such classifiers, including medical screening, genomics, and machine olfaction. Among these, machine olfaction methods have been directly inspired by the mammalian and insect olfactory systems—highly structured and well-studied biological networks that learn rapidly and non-iteratively, utilize local learning rules, resist catastrophic forgetting, can identify and learn new classes of odors (i.e., that do not map to existing representations), and can robustly identify signals of interest in the presence of strong interference. We studied the mammalian olfactory system in order to extract computational principles and algorithms that could underlie its unmatched ability to identify and classify genuinely high-dimensional signals under a variety of challenging conditions.

Most current research effort in machine olfaction is devoted to sensor development, including technologies such as multi-chamber metal oxide semiconductor (MOS) sensors (Gonzalez et al., 2011), high-density polymer sensors (Beccherelli et al., 2010), molecularly imprinted MOS and polymer sensors (Shi et al., 1999; Iskierko et al., 2016; Zhang et al., 2017), and surface acoustic wave sensors (Länge et al., 2008). In an effort to mimic properties of the biological system, there even have been efforts to develop sensors based on G protein-coupled receptor proteins bound to carbon nanotube transistors (Liu et al., 2006). In contrast, there has been relatively little effort spent mining the post-sensory networks of the olfactory system for clues to its unmatched performance, despite a broad understanding that biological odorant receptors are neither particularly specific nor particularly sensitive to odor stimuli. Rather, the power of the biological olfactory system derives from the concerted effects of the large numbers and diversity of its sensors, and by its post-sensory signal processing in the olfactory bulb and related cortices. These core principles inform recent developments in neuromorphic olfaction (Persaud et al., 2013; Schmuker et al., 2015), and have been highlighted in contemporary artificial systems work based on the similarly-structured olfactory system of insects (Schmuker et al., 2014; Mehta et al., 2017; Diamond et al., 2019).

We here present a spiking neural network (SNN)-based online learning algorithm, based on principles and motifs derived from the mammalian olfactory system, that can accurately classify noisy high-dimensional signals into categories that have been dynamically defined by few-shot learning. In order to better interpret the basis for the algorithm's capabilities, the present work focuses entirely on the properties of the first feedforward projection, omitting the spike timing-based feedback loop that forms the core network of the full OB model (Imam and Cleland, 2019). Glomerular-layer processing is represented here by two preprocessing algorithms, whereas plasticity for rapid learning is embedded in subsequent processing by the external plexiform layer (EPL) network. Information in the EPL network is mediated by patterns of spike timing with respect to a common clock corresponding to the biological gamma rhythm, and learning is based on localized spike timing-based synaptic plasticity rules. The algorithm is implemented in PyTorch for GPU computation, but designed for later implementation on state-of-the-art neuromorphic computing hardware (Davies et al., 2018); the initial version of the complete attractor model has been implemented on Intel Loihi (Imam and Cleland, 2019). We here demonstrate the interim performance of the feedforward algorithm using a well-established machine olfaction dataset with distinct challenges including multiple odorant classes, variable stimulus concentrations, physically degraded sensors, and substantial sensor drift over time.

CORE PRINCIPLES

The network is based on the architecture of the mammalian olfactory bulb (reviewed in Cleland, 2014; Nagayama et al., 2014). Primary olfactory sensory neurons (OSNs) express a single odorant receptor type from a family of hundreds (depending on animal species). The axons of OSNs that express the same receptor type converge to a common location on the surface of the olfactory bulb (OB), forming a mass of neuropil called a glomerulus. Each glomerulus thus is associated with exactly one receptor type, and serves as the basis for an OB *column*. The profile of glomerular activation levels across the hundreds of receptor types (∼400 in humans, ∼1,200 in rats and mice) that are activated by a given odorant constitutes a high-dimensional vector of sensory input (Zaidi et al., 2013). Within this first (*glomerular*) layer of the OB, a number of preprocessing computations also are performed, including a high-dimensional form of contrast enhancement (Cleland and Sethupathy, 2006) and an intricate set of computations mediating a type of global feedback normalization that enables concentration tolerance (Cleland et al., 2012). The cellular and synaptic properties of this layer also begin the process of transforming stationary input vectors into spike timing-based representations discretized by 30–80 Hz gamma oscillations (Kashiwadani et al., 1999; Li and Cleland, 2017). The EPL, which constitutes the deeper computational layer of the OB, comprises a matrix of reciprocal interactions between principal neurons activated by sensory input (mitral cells; MCs) and inhibitory interneurons (granule cells; GCs). Computations in this layer depend on fine-timescale spike timing (Lepousez and Lledo, 2013) and odor learning (Lepousez et al., 2014; Mandairon et al., 2018), and modify the information exported from the OB to its follower cortices.

Chemical sensing in machine olfaction is similarly based upon combinatorial coding (Persaud and Dodd, 1982); specificity is achieved by combining the responses of many poorly-selective sensors. In the present algorithm, networks were defined with a number of columns such that each column received input from one type of sensor in the connected input array. Columns each comprised one external tufted (ET) cell and one periglomerular (PG) cell to mediate glomerular-layer preprocessing, and one MC and a variable number of GCs to mediate EPL odorant learning and classification (**Figure 1**; see section Online Learning). Sensory input was preprocessed by the ET and PG cells of the glomerular layer (for concentration tolerance), and then delivered as excitation to the array of MCs, which generated action potentials. Each MC synaptically excited a number of randomly determined GCs drawn from across the entire network, whereas activated GCs synaptically inhibited the MC in their home column. Importantly, for present purposes, these inhibitory feedback weights were all reduced to zero to disable the feedback loop and EPL attractor dynamics, enabling study of the initial feedforward transformation based on excitatory synaptic plasticity alone. During learning, the excitatory synapses followed a STDP rule that systematically altered their weights, thereby modifying the complex receptive fields of recipient GCs in the service of odor learning. In the present study, in lieu of the modified spike timing of the MC ensemble that characterizes the output of the full model (Imam and Cleland, 2019), the binary vector describing GC ensemble activity in response to odor stimulation (0: non-spiking GC; 1: spiking GC) served as the processed data for classification. Because we here report the capacities of the initial feedforward projection of preprocessed data onto the GC interneuron array within the EPL—an initial transformation that sets the stage for ongoing dynamics not discussed herein—we refer to our present method as the *EPLff* network algorithm.

MATERIALS AND METHODS

Data Preprocessing

Sensor Scaling

We defined a set of preprocessing algorithms, any or all of which could be applied to a given data set to prepare it for efficient analysis by the core algorithm. The first of these,

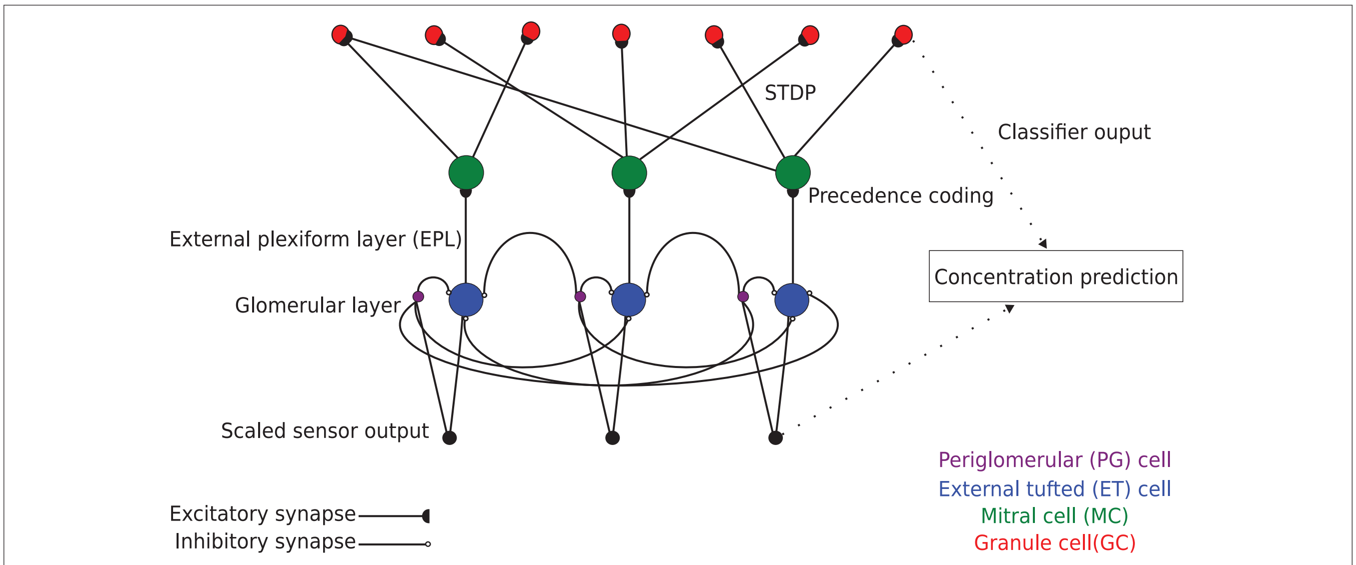


FIGURE 1 | Schematic model of *EPLff* network circuitry (three columns depicted). Sensor-scaled input data are presented in parallel to excitatory external tufted (ET) cells and inhibitory periglomerular (PG) cells in the glomerular layer. This glomerular-layer circuit performs an unsupervised concentration tolerance preprocessor step based on the graded inhibition of ET cells by PG cells. The concentration-normalized ET cell activity then is presented as input to their co-columnar mitral cells (MCs). In the external plexiform layer (EPL), comprising MC interactions with inhibitory granule cells (GCs), levels of sensory input are encoded in MCs as a spike time precedence code across the MC population. MCs project randomly onto GCs with a connection probability of 0.4. These synaptic connections are plastic, following a standard STDP rule that enables GCs to learn high-order receptive fields (Linster and Cleland, 2010). The GC population consequently learns to recognize specific odorants by measuring the similarity of high dimensional GC activity vectors with the Hamming distance metric.

sensor scaling, is applied to compensate for heterogeneity in the scales of different sensors—for example, an array comprising a combination of 1.8V and 5V sensors. One simple solution is to scale the responses of each sensor by the maximum response of that sensor. Let $x_1,\ x_2, x_3, ..., x_n$ be the responses of n sensors to a given odor and $s_1, s_2, s_3, ..., s_n$ be the maximum response values of those sensors. Then, $\frac{x_1}{s_1}, \frac{x_2}{s_2}, \frac{x_3}{s_3}, ..., \frac{x_n}{s_n}$ represent the sensor-scaled responses. The maximum sensor response vector S could be predetermined (as in sensor voltages), or estimated using a model validation set. Here, we defined S using the model validation set (10% of Batch 1 data; see section Dataset) and utilized the same value of S for scaling all subsequent learning and inference data (see section Sensor Drift). This preprocessing algorithm becomes particularly useful when analyzing data from arbitrary or uncharacterized sensors, or from arrays of sensors that have degraded and drifted non-uniformly over time.

Unsupervised Concentration Tolerance

Concentration tolerance is a critical feature of mammalian as well as insect olfaction (Cleland and Sethupathy, 2006; Cleland et al., 2012; Serrano et al., 2013). Changes in odorant concentration evoke non-linear effects in receptor activation patterns that are substantial in magnitude and often indistinguishable from those based on changes in odor quality. Distinguishing concentration differences from genuine quality differences appears to rely upon multiple coordinated mechanisms within olfactory bulb circuitry (Cleland et al., 2012), but the most important of these is a global inhibitory feedback mechanism instantiated in the deep glomerular layer (Cleland et al., 2007; Banerjee et al., 2015). The consequence of this circuit is that MC spike rates are not strongly or uniformly affected by concentration changes, and the overall activation of the olfactory bulb network remains relatively stable. We implemented this concentration tolerance mechanism as the graded inhibition of external tufted cells (ET) by periglomerular cell (PG) interneurons in the OB glomerular layer (**Figure 1**)—a mechanism based upon recent experimental findings in which ET cells serve as the primary gates of MC activation (Gire et al., 2012; Banerjee et al., 2015)—and tested its importance empirically on machine olfaction data sets. This concentration tolerance mechanism facilitates recognition of odor stimuli even when they are encountered at concentrations on which the network has not been trained; moreover, once an odor has been identified, its concentration can be estimated based on the level of feedback that the network delivers in response to its presentation. This preprocessing step requires no information about input data labels, and greatly facilitates few-shot learning.

Input from each sensor was delivered directly to PG and ET interneurons associated with the column corresponding to that sensor, and the resulting PG cell activity was delivered via graded synaptic inhibition onto all ET cells within all columns in the network. ET cells in turn then synaptically excited their corresponding, cocolumnar MCs (**Figure 1**). The approximate outcome of this preprocessor algorithm is as follows: given that $x_1^{ET}, x_2^{ET}, x_3^{ET}, ..., x_n^{ET}$ denote the responses of ET cells to odor inputs (prior to their inhibition by PG cells), and $x_1^{pg}, x_2^{pg}, x_3^{pg}, ..., x_n^{pg}$ denote the analogous responses of

PG interneurons to these same inputs, the resulting input to MC somata from ET cells following their PG-mediated lateral inhibition will be

$$\frac{x_1^{ET}}{\sum x^{pg}}, \frac{x_2^{ET}}{\sum x^{pg}}, \frac{x_3^{ET}}{\sum x^{pg}}, \ldots, \frac{x_n^{ET}}{\sum x^{pg}} \quad (1)$$

A version of this algorithm has been implemented using spiking networks on IBM TrueNorth neuromorphic hardware (Imam et al., 2012).

Core Algorithm

Cellular and Synaptic Models

We modeled the MCs and GCs as leaky integrate-and-fire neurons with an update period of 0.01 ms. The evolution of the membrane potential v of MCs and GCs over time was described as

$$\tau \frac{dv}{dt} = -v + IR \quad (2)$$

where $\tau = r_m c_m$ was the membrane time constant and r_m and c_m denote the membrane resistance and capacitance, respectively. For MCs, the input current I corresponded to sensory input received from ET cells (after preprocessing by the ET and PG neurons of the glomerular layer; **Figure 1**), whereas for GCs, I constituted the total synaptic input from convergent presynaptic MCs. In GCs, the parameter R was set to equal r_m, whereas in MCs it was set to r_m/r_{shunt}, where r_{shunt} was the oscillatory shunting inhibition of the gamma clock (described below). When $v \geq v_{th}$, where v_{th} denotes the spike threshold, a spike event was generated and v was reset to 0. The total excitatory current to GCs was modeled as

$$I = g_w(E_n - v) \quad (3)$$

where E_n was the Nernst potential of the excitatory current ($+70mv$), v was the GC membrane potential, and $g_w = \sum_{i=1}^{n} w_i g_{\max} \frac{\tau_1 \tau_2}{\tau_1 - \tau_2}(e^{\frac{-(t-t_i)}{\tau_1}} - e^{\frac{-(t-t_i)}{\tau_2}})$ describes the open probability of the AMPA-like synaptic conductances. Here, t_i denotes presynaptic spike timing, w_i denotes the synaptic weight, and $g_{\max}$ is a scaling factor.

The parameters $c_m, r_m, r_{shunt}, E_n, g_{\max}, \tau_1,$ and τ_2 were determined only once each for MCs and GCs using a synthetic data set (Borthakur and Cleland, 2017) and remained unchanged during the application of the algorithm to real datasets. The value of $w_i$at each synapse also was set to a fixed starting value based on synthetic data, but was dynamically updated according to the STDP learning rule. The spiking thresholds v_{th} of MCs and GCs were determined by assessing algorithm performance on the training and validation sets. Because we observed that using heterogeneous values of v_{th} across GCs improved performance, the values of v_{th} were randomly assigned across GCs from a uniform distribution.

Gamma Clock and Spike Precedence Code

Oscillations in the local field potential are observed throughout the brain, arising from the synchronization of activity in neuronal ensembles. In the OB, gamma-band (30–80 Hz) oscillations are associated with the coordinated periodic inhibition of MCs by GCs (Li and Cleland, 2017; Peace et al., 2017) that constrains MC spike timing (Kashiwadani et al., 1999), thereby serving as a common clock. For this work, we modeled a single cycle gamma oscillation as a sinusoidal shunting inhibition r_{shunt} delivered onto all MCs,

$$r_{shunt} = -3.8^* \cos(\frac{2\pi^* f^* t}{1000}) + 5 \quad (4)$$

where f is the oscillation frequency (40 Hz) and t is the simulation time. We used a spike precedence coding scheme for MCs (Panzeri et al., 2010) where earlier MC spike phases correspond to stronger sensor input and are correspondingly more effective at growing and maintaining spike timing-dependent plastic synapses (Linster and Cleland, 2010). In the full model, the gamma clock serves as the iterative basis for the attractor; for present purposes in the *EPLff* context it served only to structure the spike times of active MCs converging onto particular GCs (precedence coding), and thereby to govern the changes in excitatory synaptic weights according to the STDP rule (see below).

Connection Topology

MC lateral dendrites support action potential propagation to GCs across the entire extent of the OB (Xiong and Chen, 2002; Peace et al., 2017), whereas inhibition of MCs by GCs is more localized. Excitatory MC-GC synapses were initialized with a uniformly distributed random probability cp of connection and a uniform weight w_0; synaptic weights were modified thereafter by learning. The initial connection probability cp was determined using a synthetic data set (Borthakur and Cleland, 2017), and was set to $cp = 0.4$ in the present simulations. For present purposes, as noted above, GC-MC inhibitory weights were set to zero to disable attractor dynamics.

Spike Timing-Dependent Plasticity Rule

We used a modified spike timing-dependent plasticity rule (STDP; Song et al., 2000; Dan and Poo, 2004) to regulate MC-GC excitatory synaptic weight modification. Briefly, synaptic weight changes were initiated by GC spikes and depended exponentially upon the spike timing difference between the postsynaptic GC spike and the presynaptic MC spike. When a presynaptic MC spike preceded its postsynaptic GC spike within the same gamma cycle, w for that synapse was increased; in contrast, when MC spikes followed GC spikes, or when a GC spike occurred without a presynaptic MC spike, w was decremented. Synaptic weights were limited by a maximum weight $w_{\max}$. The pairing of STDP with MC spike precedence coding discretized by the gamma clock generated a k *winners take all* rule, in which the value of k depended substantially on the GC spike threshold v_{th} and the maximum excitatory synaptic weight $w_{\max}$. Under this rule, activated GCs were transformed from non-specialized cells receiving weak inputs from a broad and random distribution of MCs into specialized, fully differentiated neurons that responded only to coordinated activation across a specific ensemble of k MCs. Under all training conditions, for present purposes, we set a high learning rate such that, after one cycle of learning, each of the synapses could have one of only three values: w_0, $w_{\max}$, or 0.

The STDP parameters were similar to our previous work using a synthetic data set (Borthakur and Cleland, 2017); among these, only the maximum synaptic weight $w_{\max}$ was tuned based on validation set performance. For this feedforward implementation, online learning without the requirement of storing training data yielded its best validation set performance when $w_{\max} = w_0$, such that learning was limited to long-term synaptic depression (Borthakur and Cleland, 2017).

Classification

For the classification of test odorants in this reduced feedforward *EPLff* implementation, we calculated the Hamming distance between the binary vectors of GC odorant representations. Specifically, for every input, GCs generated a binary vector based upon whether the GC spiked (1) or did not spike (0). We matched the similarity of test set binary vectors with the training set vector(s) using the Hamming distance and classified the test sample based upon the label of the closest training sample. Alternatively, an overlap metric between GC activation patterns also was calculated (Equation 6 from Linster and Cleland, 2010); results based on this method were reliably identical to those of the Hamming distance and hence were omitted from this report. Classification was set to *none of the above* if the Hamming distance of the GC binary vectors was >0.5, or if the overlap metric was <0.5.

Dataset

We tested our algorithm on the publicly available UCSD gas sensor drift dataset (Vergara et al., 2012; Rodriguez-Lujan et al., 2014), slightly reorganized to better demonstrate online learning. The original dataset contains 13,910 measurements from an array of 16 polymer chemosensors exposed to six gas-phase odorants spanning a wide range of concentrations (10–1,000 *ppmv*) and distributed across 10 batches that were sampled over a period of 3 years to emphasize the challenge of sensor drift over time (**Table 1**). Owing to drift, the sensors' output statistics change drastically over the course of the 10 batches; between this property, the six different gas types, and the wide range of concentrations delivered, this dataset is well-suited to test the capabilities of the present algorithm without exceeding the learning capacity of its feedforward architecture (**Figure 1**). For the online learning scenario, we sorted each batch of data according to the odorant trained, but did not organize the data according to concentration. Hence, each training set comprised 1–10 odorant stimuli of the same type but at randomly selected concentrations. Test sets always included all six different odorants, again at randomly selected concentrations. For sensor scaling and the fine-tuning of the algorithm, we used 10% of the Batch 1 data as a validation set. The six odorants in the dataset are, in the order of training used herein: ammonia, acetaldehyde, acetone, ethylene, ethanol, and toluene. Batches 3–5 included only five different odorant stimuli, omitting toluene.

Eight features per chemosensor were recorded in the UCSD dataset, yielding a 128-dimensional feature vector. However, in contrast to previous efforts (Liu et al., 2015; Zhang and Zhang, 2015; Yan et al., 2017; Ma et al., 2018), we chose to use only one feature per sensor in our analysis (the steady state response level), for a total of 16 features. We imposed this restriction to challenge our algorithm, and because generating features from raw data requires additional processing, energy and time, all of which can impair the effectiveness of field-deployable hardware (Yin et al., 2018). Importantly, however, the sensor scaling and concentration tolerance preprocessors described above (section Data Preprocessing) would enable the *EPLff* network to utilize the full 128-dimensional dataset without specific adaptations other than expanding the number of columns accordingly.

RESULTS

Data Preprocessing

All sensory input data were preprocessed before being presented to the network. First, sensor scaling was applied to weight the 16 sensors equally in subsequent computations. The mean raw responses of the 16 sensors differed widely, with some sensors exhibiting an order of magnitude greater variance than others across the 10 odorants tested (**Figure 2A**). Sensor scaling (**Figure 2B**) mitigated this effect by scaling each sensor's gain such that the dynamic ranges of all sensors across the test battery were effectively equal. This process enabled each sensor to contribute a comparable amount of information to subsequent computations (up to a limit imposed by each sensor's signal to noise ratio), and improved network performance by maintaining consistent mean activity levels across test odorants.

Since each odorant was presented at a wide range of randomly selected concentrations, the response of the sensor array to a given odorant varied widely across presentations (most clearly observable in **Figure 2B**). Application of the unsupervised concentration tolerance preprocessor sharply and selectively reduced the concentration-specific variance among responses to presented odorants (**Figure 2C**). These preprocessed odorant signatures then were presented to the plastic *EPLff* network for training or classification. Notably, this preprocessor step greatly facilitated cross-concentration odorant recognition, even enabling the accurate classification of samples presented at concentrations that were not included in the training set. This was particularly important for one- and few-shot learning, in which the network was trained on just one or a few exemplars (respectively), at unknown concentration(s), such that most of the odorants in the test set were presented at concentrations on which the network had never been trained.

The sensor scaling preprocessor (retaining the scaling factors determined from the 10% validation set of Batch 1), combined with the normalization effects of the subsequent concentration tolerance preprocessor, had the additional benefit of restoring the dynamic range of degraded sensors in order to better match classifier network parameters. Because of this, the network did not need to be reparameterized to effectively analyze the responses of the degraded sensors in the later batches of this dataset. Compared to the raw sensor output of Batch 1 (**Figure 2A**; collected from new sensors), the raw sensor output of Batch 7 (**Figure 2D**; collected after 21 months of sensor deterioration) was reduced to roughly a third of its original range.

TABLE 1 | Properties of the UCSD gas sensor drift dataset.

	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Batch 6	Batch 7	Batch 8	Batch 9	Batch 10
Months	1–2	3–10	11–13	14–15	16	17–20	21	22–23	24–30	36
#Samples	445	1,244	1,586	161	197	2,300	3,613	294	470	3,600

Months denotes the age of the sensor array during the sampling of the corresponding dataset. #Samples denotes the number of samples provided by the dataset in that particular batch.

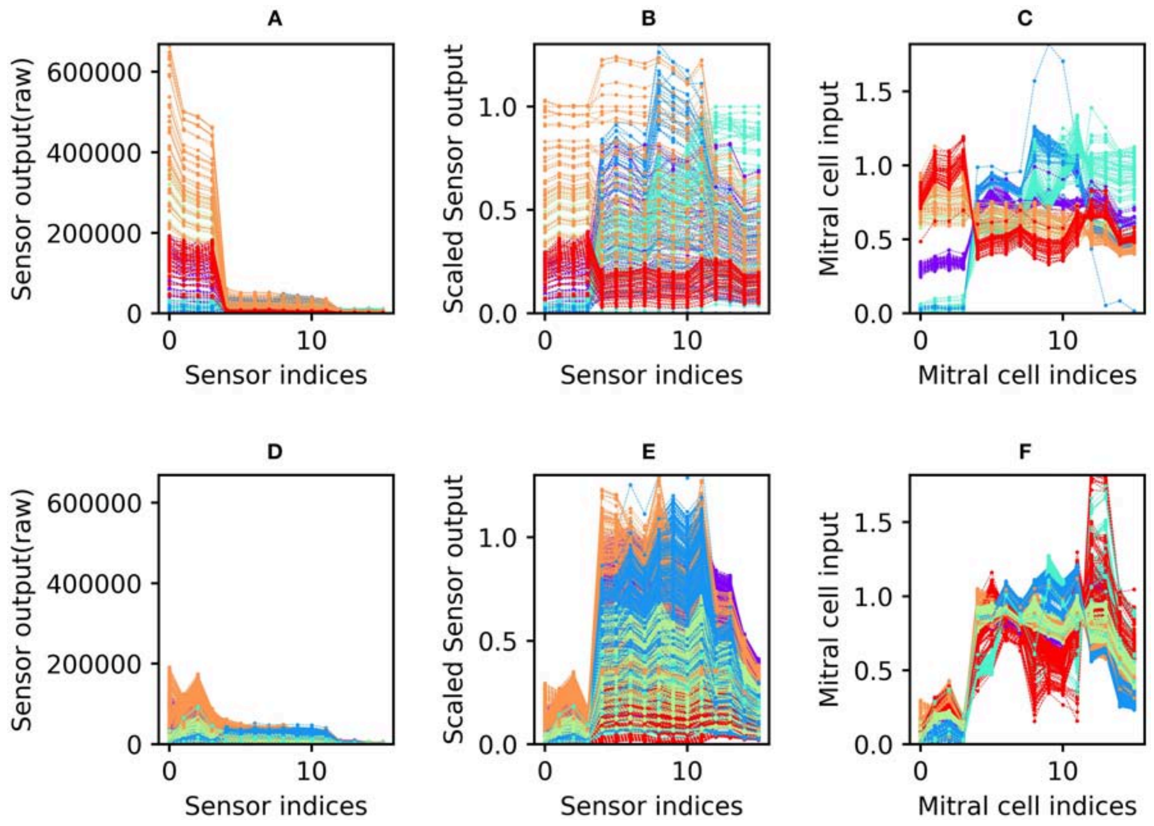


FIGURE 2 | Sensor drift and the application of the sensor scaling and concentration tolerance preprocessors. **(A)** Raw sensory data from the Batch 1 training set. The abscissa denotes the 16 different sensors of the array; the ordinate denotes the magnitude of their responses to specific odorants. The six different colors denote the six odorants of the dataset battery (ammonia, *purple*; acetaldehyde, *blue*; acetone, *aqua*; ethylene, *green*; ethanol, *orange*; and toluene, *red*). Note that each odorant is presented at many different concentrations (Vergara et al., 2012). **(B)** Sensory input from the Batch 1 data shown in **(A)** after preprocessing for sensor scaling. The absolute range of output values is now rendered consistent across all of the sensors in the array. **(C)** Sensory input from the same Batch 1 data after subsequent preprocessing for concentration tolerance by glomerular layer circuitry (**Figure 1**, ET and PG). The sensory signatures of each of the six odors are now more internally consistent, with less variance owing to the concentration differences inherent in the original data **(D–F)**. As **(A–C)** but with Batch 7 training data. These data were taken from the same set of sensors as depicted in **(A–C)**, but after 21 months of operational degradation, including intermittent periods of use and disuse (**Table 1**).

Sensor scaling (**Figure 2E**) mitigated this effect by magnifying sensor responses into the dynamic range expected by the network. Subsequent preprocessing for concentration tolerance effectively reduced concentration-specific variance, revealing a set of odorant profiles (**Figure 2F**) that, while qualitatively dissimilar to their profiles based on the same sensors 21 months prior (**Figure 2C**), appear only modestly degraded in terms of their distinctiveness from one another.

For many machine olfaction applications, it is useful to estimate the concentrations of gases in the vicinity of the sensors. We sought to use the information extracted from the concentration tolerance preprocessor to estimate the concentrations of test samples after classification. The concentration estimation curve was a function of both odorant identity and the total sensor response profile. Using the sum of the 16 sensor responses (S), we fitted an odorant-specific quadratic curve for an implicit model of response profiles across concentrations $C: C = ax^2 + b$, where the parameters a and b were determined from the training set. **Figure 3** illustrates total sensor responses across concentrations compared to this theoretical prediction for all six odorant gases in Batches 1 and 7. The mean absolute error (MAE) of the prediction (in *ppmv*)

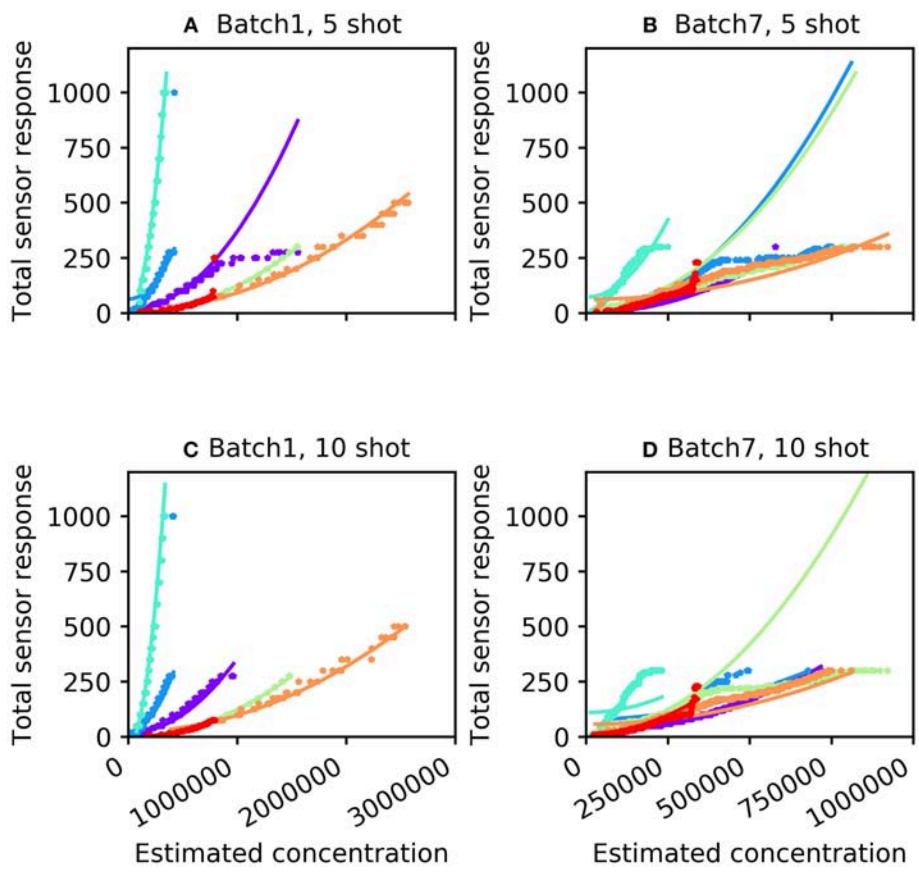


FIGURE 3 | Concentration response function predicted by the algorithm (curves) compared with measured sensor responses across multiple concentrations (stars). **(A)** Batch 1 data with five-shot training. **(B)** Batch 7 data with five-shot training. **(C)** Batch 1 data with 10-shot training. **(D)** Batch 7 data with 10-shot training. The colors denoting particular odorants are the same as in **Figure 2**.

TABLE 2 | Concentration estimation performance on test sets of all batches of UCSD gas sensor drift dataset for 5- and 10-shot learning (see **Figure 3**).

	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Batch 6	Batch 7	Batch 8	Batch 9	Batch 10	Mean error
5 shot	35.14	51.39	32.00	44.73	66.02	37.01	76.60	17.89	0.34	71.06	43.22
10 shot	23.35	33.60	28.97	35.75	64.71	25.23	58.18	19.66	0.84	52.61	34.29

Concentrations were estimated using the predicted labels and raw sensor input. Errors represent experimental deviation from the predicted quadratic concentration curves and are in units of ppmv.

was estimated as

$$\frac{\sum_{n} \left|C_{pred} - C_{actual}\right|}{n} \tag{5}$$

where n denotes the total number of samples. For the five-shot training of Batch 1 (i.e., five random samples drawn from Batch 1 for each odorant), the MAE was 35.14 units (**Table 2**). This error was reduced to 23.35 for 10-shot learning (**Table 2**). Similarly, the MAE for Batch 7 decreased from 76.60 (five-shot) to 58.18 (10-shot). To the best of our knowledge, this is the first parallel network architecture to provide an estimate of concentration along with concentration tolerance.

Online Learning

Unlike biological odor learning, artificial neural networks optimized for a certain task tend to suffer from catastrophic forgetting, and the pursuit of online learning capabilities in deep networks is a subject of active study (McCloskey and Cohen, 1989; Kemker and Kanan, 2017; Kirkpatrick et al., 2017; Velez and Clune, 2017; Zenke et al., 2017; Serrà et al., 2018). In contrast, the *EPLff* learning network described herein naturally resists catastrophic forgetting, exhibiting powerful online learning using a fast spike timing-based coding metric. Moreover, we include a *none of the above* outcome which permits classification only above a threshold level of confidence (Huerta and Nowotny, 2009). Hence, after being trained on one odorant, the network could identify a test sample as either that odorant or *none of the above*. After subsequently training the network on a second odorant, it could classify a test sample as either the first trained odorant, the second trained odorant, or *none of the above*. This online learning capacity enables *ad hoc* training of the network, with intermittent testing if desired, with no need to train on or even establish the full list of classifiable odorants in advance. It also facilitates training under missing data conditions (e.g., batches 3–5 contain samples from only five odorants, unlike the other batches which include six odorants), and could be utilized to trigger new learning in an unsupervised exploration context. Finally, once learned, the training set data need not be stored.

To analyze the 16-sensor UCSD dataset, we constructed a 16-column spiking network with 4800 GC interneurons and a uniformly random MC-GC connection probability $cp = 0.4$. This number of GCs was selected because it was the smallest network that achieved asymptotic performance on the validation dataset (Batch 1, one-shot learning; **Table 3**). We then trained this network on ammonia using 10 different few-shot training schemes: one-shot, two-shot, three-shot, up through 10-shot in order to measure the utility of additional training. Test data (across all trained odorants and all concentrations in the dataset) were classified with 100.0% accuracy in all cases (**Figure 4A**; average of three runs). We subsequently trained each of these trained networks on acetaldehyde, using the same number of training trials in each case. After one-shot learning of acetaldehyde, the network classified all trained odorants with 99.61 ± 0.28% accuracy (average of three runs). After subsequent one-shot learning of acetone, classification performance was 95.65 ± 0.19%; after ethylene, 96.06 ± 0.17%; after ethanol, 90.94 ± 0.0%, and finally, after one-shot training on the sixth and final odorant, toluene, test set classification performance across all odorants was 90.27 ± 0.12%. Multiple-shot learning generally produced correspondingly higher classification performance as the training regimen expanded (**Figure 4A**). Classification using an overlap metric (Linster and Cleland, 2010) rather than the Hamming distance yielded almost identical results (not shown). Critically, classification performance did not catastrophically decline as additional odorants were learned in series (**Figure 4**, *purple* to *red (orange)* traces in order), particularly when higher-quality sensors were used (**Figures 4A–E**) or when larger multiple-shot training sets were employed (**Figure 4**, panel abscissas). These results illustrate that the *EPLff* network, even in the absence of the full model's recurrent component, exhibits true online learning.

The availability of data in the UCSD dataset from over 3 years of sensor deterioration enabled the testing of this online learning algorithm with both fresh and degraded sensor arrays. **Figures 4B–J** presents classification results from the same procedures described above but using progressively older and more degraded sensors (Batches 2–10; **Table 1**; Vergara et al., 2012). Classification performance declined overall as the sensors deteriorated in later batches (**Figures 4F–J**), but could be substantially rescued by expanding the training regimen from one-shot to few-shot learning. Overall, multiple-shot training reliably improved classification performance, though the residual variance across different training regimes suggests that the

TABLE 3 | Effect of increased numbers of GCs in the network (GC vector length) on *EPLff* classification accuracy by the Hamming distance criterion, based on one-shot learning using the Batch 1 validation set.

#GC	1 class trained	2 classes trained	3 classes trained	4 classes trained	5 classes trained	6 classes trained
160	100.0	100.0	95.65	96.15	85.71	84.44
1,600	100.0	100.0	95.65	96.15	85.71	86.67
4,800	100.0	100.0	95.65	96.15	85.71	88.89
9,600	100.0	100.0	95.65	96.15	85.71	88.89
14,400	100.0	100.0	95.65	96.15	85.71	88.89
19,200	100.0	100.0	95.65	96.15	85.71	88.89

Connection probabilities and initial synaptic weights were consistent across all simulations.

random selection of better or poorer class exemplars for training (particularly noting the uncontrolled variable of concentration) exerted a measurable effect on performance (**Figure 4**; **Table 4**).

Batch 10 of the UCSD dataset poses a relatively challenging classification problem. To produce it, the sensors were intentionally degraded and contaminated by turning off sensor heating for 5 months following the production of Batch 9 data (Vergara et al., 2012). Prior work with this dataset has achieved up to 73.28% classification performance on Batch 10, without online learning and using a highly introspective approach tailored for this specific dataset (Yan et al., 2017). In contrast, 10-shot learning on Batch 10 using the present *EPLff* algorithm achieved 85.43% classification accuracy.

To compare the *EPLff* network's resistance to catastrophic forgetting against an existing standard method, we built a 16-input multi-layer perceptron (MLP) comprising 16 input units for raw sensor input (ReLu activation), 4,800 hidden units (ReLu activation), and six output units for odorant classification. The MLP was trained using the Adam optimizer (Kingma and Ba, 2014) with a constant learning rate of 0.001. Since there was no straightforward way of implementing *none of the above* in an MLP, the MLP was only trained using two or more odorants (**Figure 5**). After initial, interspersed training on two odorants from Batch 1, the MLP classified test odorants at high accuracy (99.41 ± 0.0%; average of three runs; **Figure 5A**). However, its classification accuracy dropped sharply after the subsequent, sequential learning of odorant 3 (30.61 ± 0.0% accuracy), odorant 4 (16.24 ± 9.29%), odorant 5 (18.13 ± 0.0%), and odorant 6 (15.99 ± 0.0%) (**Figure 5**). Catastrophic forgetting is a well-known limitation of MLPs, and is presented here simply to quantify the contrast in online learning performance between the *EPLff* implementation and a standard network of similar scale.

Online Reset Learning for Mitigating Sensor Drift

One of the most challenging problems of machine olfaction is *sensor drift*, in which the sensitivity and selectivity profiles of chemosensors gradually change over weeks to months of use or disuse. Efforts to compensate for this drift have taken many forms, from simply replacing sensors to designing highly introspective or specific corrective algorithms. For example,

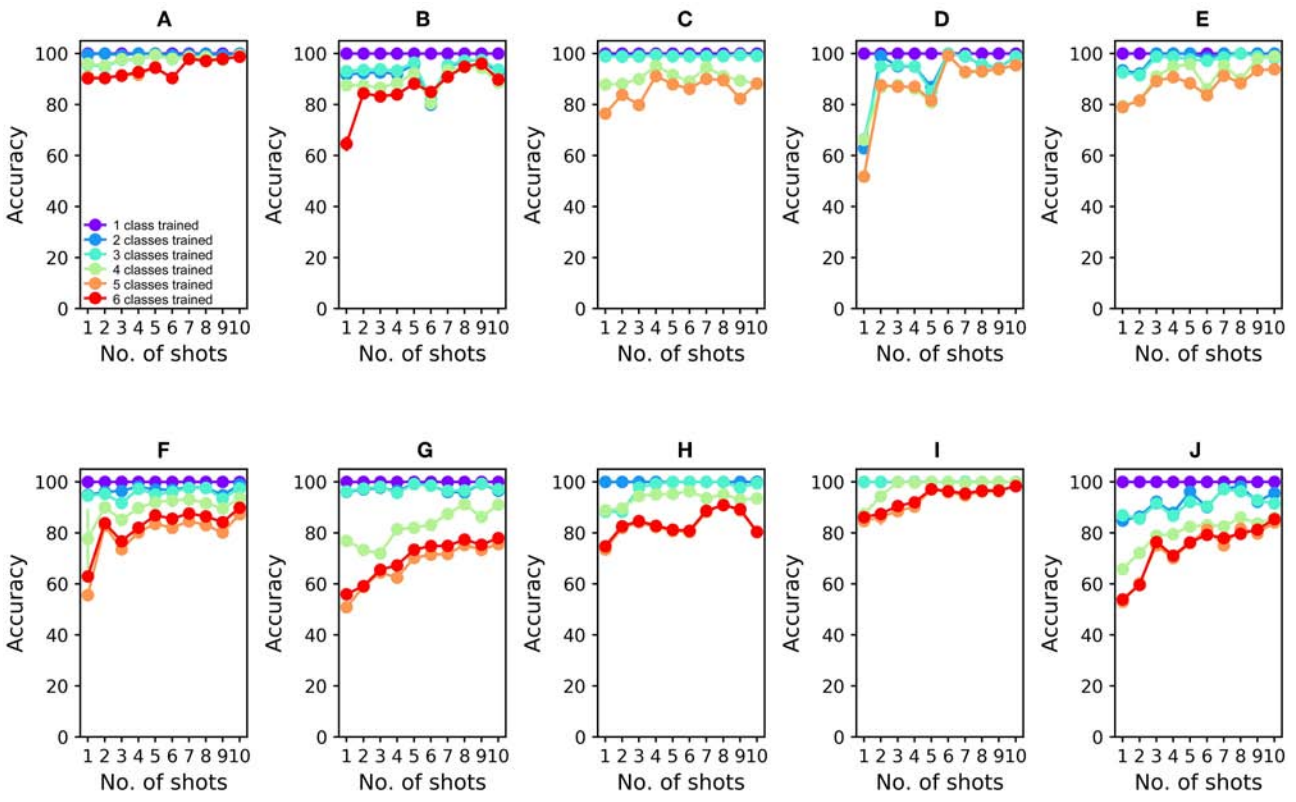


FIGURE 4 | *EPLff* algorithm performance on UCSD gas sensor drift dataset. **(A)** Classification performance during online training and testing of Batch 1 data. Plotted values depict the average classification performance for all test samples of the trained classes. Ammonia was trained first; the purple plot (*One trained class*) denotes the classification accuracy of ammonia test samples as either *ammonia* or *none of the above*. Acetaldehyde was trained second; the blue plot (*Two trained classes*) denotes the classification accuracy of trained-class samples as either *ammonia*, *acetaldehyde*, or *none of the above*. Online training proceeded with acetone (aqua), ethylene (green), ethanol (orange), and toluene (red) in that order, with the final plot (red or orange) finally denoting the average classification accuracy of all samples into one of the six (or five) odorant classes or, potentially, as *none of the above*. Classification performance degraded slightly as the number of trained odorant representations in the network increased, but improved as the number of learning shots increased **(B)**. As **(A)**, but after training and testing with Batch 2 data **(C–J)**. As **(A,B)**, but after training and testing with Batch 3–10 data, in corresponding order. The colors denoting particular odorants are consistent with **Figures 2**, **3**.

one approach requires the non-random, algorithmically guided selection of relevant samples across batches and/or the utilization of test data as unlabeled data for additional training (Zhang and Zhang, 2015; Yan et al., 2017; Ma et al., 2018). Despite some partial successes in these approaches, the real-world challenge of sensor drift is a fundamentally ill-posed problem, in which the rapidity and nature of functional drift is highly dependent on the idiosyncratic chemistry of individual sensors and specific sensor-analyte pairs.

We argue that the most practical solution to this challenge is to retrain the network as needed to maintain performance, leveraging its rapid, online learning capacity. Specifically, MC-GC synaptic weights are simply reset to their untrained values and the network then is rapidly retrained using the new (degraded) sensor response profiles (*reset learning*). Retraining is not a new approach, of course, but overtly choosing a commitment to heuristic retraining as the primary method for countering sensor drift is important, as it determines additional criteria for real-world device functionality that candidate solutions must address, such as the need for rapid, ideally online retraining in the field and potentially a tolerance for lower-fidelity training sets. Specifically, retraining a traditional classification network may require:

1. Prior knowledge of the number of possible odor classes to be identified,
2. A sufficiently large and representative training set incorporating each of these classes,
3. The retuning of network hyperparameters to match the altered characteristics of the degraded sensors, requiring an indeterminate number of training iterations.

The *EPL* network is not constrained by the above requirements. As demonstrated above, it can be rapidly retrained using small samples of whatever training sets are available and then be updated thereafter—including the subsequent introduction of new classes. The storage of training data for retraining purposes is unnecessary as the network does not suffer from catastrophic forgetting. Finally, the present network does not require hyperparameter retuning. Here, only the MC-GC weights were updated during retraining

TABLE 4 | Mean *EPLff* classification accuracies across all test odorants on the UCSD drift data set by the Hamming distance criterion.

	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Batch 6	Batch 7	Batch 8	Batch 9	Batch 10	Average
1 shot	95.42	83.62	92.48	69.43	88.89	80.97	79.37	87.59	93.04	74.11	84.49
2 shot	95.15	90.38	93.97	93.68	89.35	91.35	80.98	90.41	94.68	77.36	89.73
3 shot	96.28	89.91	93.60	93.02	95.78	87.30	82.98	93.52	96.51	85.72	91.46
4 shot	96.62	90.29	97.02	92.64	96.87	91.25	83.89	93.26	97.02	82.58	92.14
5 shot	97.99	93.47	95.54	86.94	96.40	92.42	87.35	92.89	99.05	87.19	92.92
6 shot	96.00	85.44	94.70	99.69	93.01	92.22	87.77	92.93	98.73	87.36	92.78
7 shot	98.80	94.22	96.54	96.66	97.03	93.47	87.79	95.11	98.30	88.39	94.63
8 shot	98.59	96.45	95.79	95.48	95.64	92.78	89.43	96.16	98.81	90.34	94.95
9 shot	98.39	96.92	94.11	95.35	98.06	90.37	88.92	94.84	98.82	88.32	94.41
10 shot	99.39	92.44	94.95	97.73	98.22	94.55	89.74	92.30	99.48	90.46	94.93

*Odorant-specific classification accuracies are depicted in **Figure 4**.*

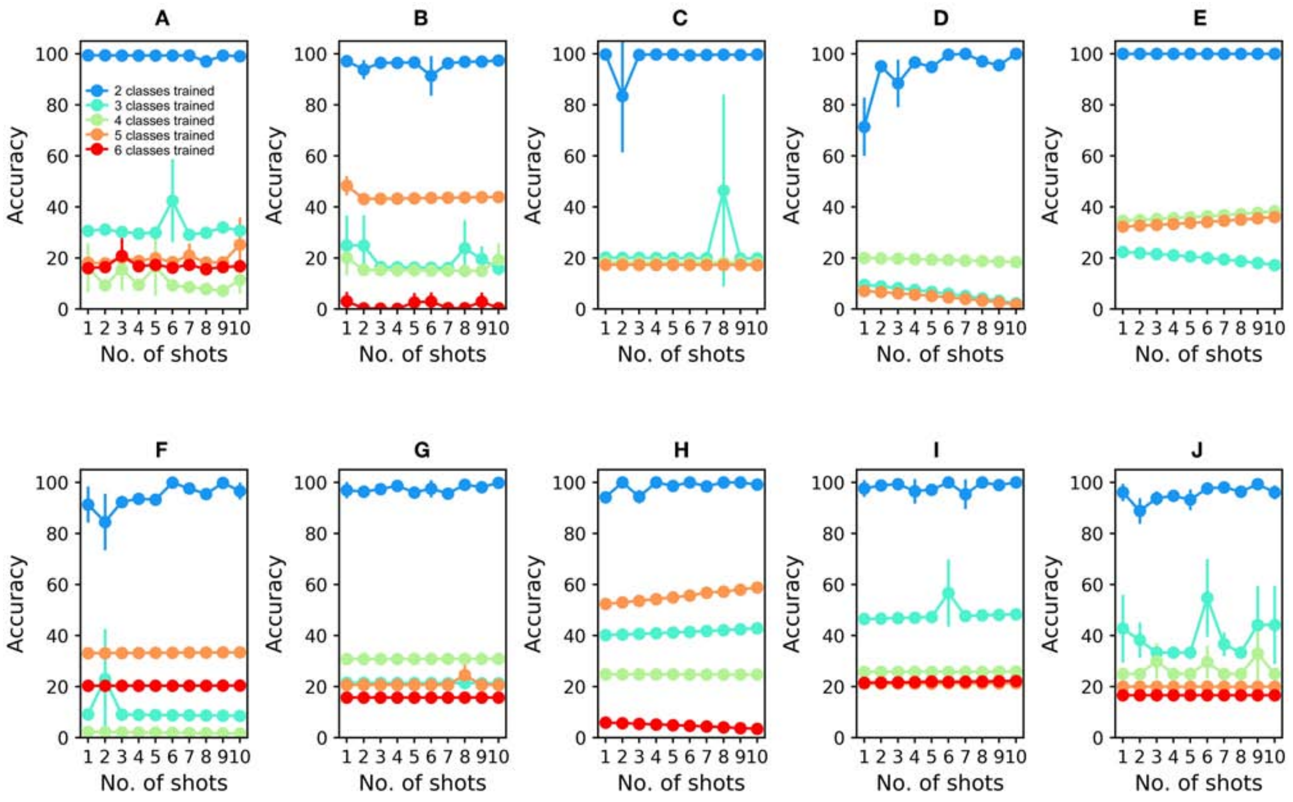


FIGURE 5 | Multilayer perceptron (MLP) performance on UCSD gas sensor drift dataset during online learning. **(A)** Classification performance during online training and testing of Batch 1 data. The network was first trained with ammonia and acetaldehyde (see text); the blue plot denotes the classification accuracy of test samples of these two odorants. Online training proceeded with acetone (aqua), ethylene (green), ethanol (orange), and toluene (red) in that order, with the final plot denoting the average classification accuracy of all samples into one of the five (or four) odorant classes. Unlike the *EPLff* algorithm, the MLP suffered catastrophic forgetting after training on new sample types. **(B–J)** MLP performance during online training and testing of Batch 2–10 data, in corresponding order. Except for the combination of ammonia and acetaldehyde in the first training set, the colors denoting particular odorants are consistent with **Figures 2–4**.

(using the same STDP rule); sensor scaling factors and all other parameters were ascertained once, using the 10% validation set of Batch 1, and held constant thereafter. Moreover, the *none of the above* classifier confidence feature facilitates awareness of when the network may require retraining; an increase in *none of the above* classifications provides an initial cue that then can be evaluated using known samples.

To assess the efficacy of this approach, we tested the *EPLff* algorithm on the UCSD dataset framed as a sensor drift problem. The procedure for this approach, and consequently the results, are identical to those of section Online Learning above (**Figure 4**;

Table 4). Importantly, the sensor scaling factors and network parameters were tuned only once, using the validation set from Batch 1, on the theory that the concept of rapid reset was incompatible with a strategy of re-optimizing multiple network hyperparameters. Hence, no parameter changes were permitted, other than the MC-GC excitatory synaptic weights that were updated normally during training according to the STDP rule (In order to avoid duplication of figures, this constraint was observed in the simulations of section Online Learning as well). As described above (**Figure 4**), Batch 1 training samples from all six odorants again were presented to the network in an online learning configuration, and classification performance then was assessed by Batch 1 test data. MC-GC synaptic weights then were reset to the default values (the *reset*), after which Batch 2 training samples were presented to the network in the same manner, followed by testing with Batch 2 test data including all odorants and concentrations. We repeated this process for batches 3–10. We also assessed post-reset classification performance across all batches based on a maximally rapid reset (i.e., one-shot learning) and compared this to performance after expanded training protocols up through 10-shot learning. All classification performance results (averaged across three full repeats each) are depicted in **Figure 4** and **Table 4**. In general, while modest increases in classification accuracy were observed when the training set size was larger, these results demonstrate scalability, showing that the *EPLff* algorithm classifies large sets of test data with reasonable accuracy even based on small training sets and lacking control over the concentrations of presented odorants.

DISCUSSION

We present a neural network algorithm that achieves superior classification performance in an online learning setting while not being specifically tuned to the statistics of any particular dataset. This property, coupled with its few-shot learning capacity and SNN architecture, renders it particularly appropriate for field-deployable devices based on learning-capable SNN hardware (Davies et al., 2018; Imam and Cleland, 2019), recognizing that the interim use of the Hamming distance for nearest-neighbor classification in the present *EPLff* framework will not be part of such a deployable system. This algorithm is inspired by the architecture of the mammalian olfactory bulb, but is comparably applicable to any high-dimensional dataset that lacks internal low-dimensional structure.

The present *EPLff* incarnation of the network utilizes one or more preprocessor algorithms to prepare data for effective learning and classification by the core network. Among these is an unsupervised concentration tolerance algorithm derived from feedback normalization models of the biological system (Cleland et al., 2007, 2012; Banerjee et al., 2015), a version of which has been previously instantiated in SNN hardware (Imam et al., 2012). Inclusion of this preprocessor enables our algorithm to quickly learn reliable representations based on few-shot learning from odorant samples presented at different and unknown concentrations. Moreover, the network then can generalize across concentrations, correctly classifying unknown test odorants presented at concentrations on which the network was never trained, and even estimating the concentrations of these unknowns.

The subsequent, plastic EPL layer of the network is based on a high-dimensional projection of sensory input data onto a network of interneurons known as granule cells (GCs). In the present feed-forward implementation, our emphasis is on the roles and capacities of two sequential preprocessor steps followed by the STDP-driven plasticity of the excitatory MC-GC synapses. Subsequent extensions of this work will restore the feedback architecture of the original model (Imam and Cleland, 2019) while enabling a more sophisticated development of learned classes within the high-dimensional projection field. Even in its present feedforward form, however, the *EPLff* algorithm exhibits (1) rapid, online learning of arbitrary sensory representations presented in arbitrary sequences, (2) generalization across concentrations, (3) robustness to substantial changes in the diversity and responsivity of sensor array input without requiring network reparameterization, and, by virtue of these properties, is capable of (4) effective adaptation to ongoing sensor drift via a rapid reset-and-retraining process termed reset learning. This capacity for fast reset learning represents a practical strategy for field-deployable devices, in which a training sample kit could be quickly employed in the field to retune and restore functionality to a device in which the sensors may have degraded. Importantly for such purposes, the *EPLff* algorithm was not, and need not be, crafted to the statistics of any particular data set, nor was the network pre-exposed to testing set data as has been done in some approaches (Zhang and Zhang, 2015; Yan et al., 2017).

Because field-deployable devices require a level of generic readiness for undetermined or underdetermined problems, and these *EPLff* properties favor such readiness, we have emphasized the portability of these algorithms to neuromorphic hardware platforms that may come to drive such devices. Interestingly, many of the features of the biological olfactory system that have inspired this design are appropriate for such devices. Spike timing and event-based algorithms are attractive candidates for compact, energy-efficient hardware implementation (Imam et al., 2012; Merolla et al., 2014; Qiao et al., 2015; Diehl et al., 2016; Esser et al., 2016; Davies et al., 2018). Spike timing metrics can compute similar transformations as analog and rate-based representations; indeed, it has been proposed that spike based computations could in principle exhibit all of the computational power of a universal Turing machine (Maass, 1996, 2015). STDP is a localized learning algorithm that is highly compatible with the colocalization of memory and compute principle of neuromorphic design, and its theoretical capacities have been thoroughly explored in diverse relevant contexts (Nessler et al., 2009; Linster and Cleland, 2010; Schmiedt et al., 2010; Bengio et al., 2015; O'Connor et al., 2018). Our biologically constrained approach to algorithm design also provides a unified and empirically verified framework to investigate the interactions of these various algorithms and information metrics, to better interpret and apply them to artificial network design.

Other groups have previously proposed networks for gas sensor data analysis inspired by biological olfactory systems. Models of olfactory bulb and piriform cortical activity have been applied to analyze chemosensor array data (Raman and Gutierrez-Osuna, 2005; Raman et al., 2006). Algorithms based on the insect olfactory system have been employed to learn and identify odor-like inputs (Diamond et al., 2016; Delahunt et al., 2018) as well as to identify handwritten digits—visual inputs incorporating additional low-dimensional structure (Huerta and Nowotny, 2009; Delahunt and Kutz, 2018; Diamond et al., 2019). More broadly, insect mushroom bodies in particular have been deeply studied in terms of both their pattern separation and associative learning capacities (Hige, 2018; Cayco-Gajic and Silver, 2019). These capacities potentiate one another in service to odor learning and the classification of learned odor-like signals, though they also have been applied to more complex tasks (Ardin et al., 2016; Peng and Chittka, 2017). In the present work, we sought to design artificial learning networks to replicate some of the most powerful capabilities of the biological olfactory system, in particular its capacity for rapid online learning and the fast and effective classification of learned odorants despite ongoing changes in sensor properties and the unpredictability of odor concentrations. Future work will extend this framework to incorporate the feedback dynamics of the biological system, increase the dimensionality of sensor arrays, and develop more sophisticated biomimetic classifiers.

AUTHOR CONTRIBUTIONS

TC originally conceived the algorithm, which was vetted and modified for present purposes by AB and TC. AB designed, programmed, and performed the simulations. AB and TC designed the figures and wrote the paper.

FUNDING

This work was supported by a Cornell University Sage fellowship to AB and an Intel Neuromorphic Research Community faculty award and NIH/NIDCD awards DC014367 and DC014701 to TC.

ACKNOWLEDGMENTS

The authors acknowledge Dr. Nabil Imam for interesting discussions regarding the *EPLff* algorithm, and Dr. Ramon Huerta and Dr. Jordi Fonollosa for discussions regarding the UCSD Gas Sensor Array Drift Dataset.

REFERENCES

Ardin, P., Peng, F., Mangan, M., Lagogiannis, K., and Webb, B. (2016). Using an insect mushroom body circuit to encode route memory in complex natural environments. *PLoS Comput. Biol.* 12:e1004683. doi: 10.1371/journal.pcbi.1004683

Banerjee, A., Marbach, F., Anselmi, F., Koh, M. S., Davis, M. B., Garcia da Silva, P., et al. (2015). An interglomerular circuit gates glomerular output and implements gain control in the mouse olfactory bulb. *Neuron* 87, 193–207. doi: 10.1016/j.neuron.2015.06.019

Beccherelli, R., Zampetti, E., Pantalei, S., Bernabei, M., and Persaud, K. C. (2010). Design of a very large chemical sensor system for mimicking biological olfaction. *Sens. Actuators B Chem.* 146, 446–452. doi: 10.1016/j.snb.2009.11.031

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv:1502.04156* [cs]. Available online at: http://arxiv.org/abs/1502.04156 (accessed May 13, 2018).

Borthakur, A., and Cleland, T. A. (2017). "A neuromorphic transfer learning algorithm for orthogonalizing highly overlapping sensor array responses," in *2017 ISOCS/IEEE International Symposium on Olfaction and Electronic Nose (ISOEN)* (Montreal, QC), 1–3.

Cayco-Gajic, N. A., and Silver, R. A. (2019). Re-evaluating circuit mechanisms underlying pattern separation. *Neuron* 101, 584–602. doi: 10.1016/j.neuron.2019.01.044

Cleland, T. A. (2014). Construction of odor representations by olfactory bulb microcircuits. *Prog. Brain Res.* 208, 177–203. doi: 10.1016/B978-0-444-63350-7.00007-3

Cleland, T. A., Chen, S.-Y. T., Hozer, K. W., Ukatu, H. N., Wong, K. J., and Zheng, F. (2012). Sequential mechanisms underlying concentration invariance in biological olfaction. *Front. Neuroeng.* 4:21.

Cleland, T. A., Johnson, B. A., Leon, M., and Linster, C. (2007). Relational representation in the olfactory system. *Proc. Natl. Acad. Sci. U.S.A.* 104, 1953–1958. doi: 10.1073/pnas.0608564104

Cleland, T. A., and Sethupathy, P. (2006). Non-topographical contrast enhancement in the olfactory bulb. *BMC Neurosci.* 7:7. doi: 10.1186/1471-2202-7-7

Dan, Y., and Poo, M.-M. (2004). Spike timing-dependent plasticity of neural circuits. *Neuron* 44, 23–30. doi: 10.1016/j.neuron.2004.09.007

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Delahunt, C. B., and Kutz, J. N. (2018). *A Moth Brain Learns to Read MNIST*. Available online at: https://openreview.net/forum?id=HyYuqoCUz (accessed May 13, 2018).

Delahunt, C. B., Riffell, J. A., and Kutz, J. N. (2018). Biological mechanisms for learning: a computational model of olfactory learning in the *Manduca sexta* moth, with applications to neural nets. *Front. Comput. Neurosci.* 12:102. doi: 10.3389/fncom.2018.00102

Diamond, A., Schmuker, M., Berna, A. Z., Trowell, S., and Nowotny, T. (2016). Classifying continuous, real-time e-nose sensor data using a bio-inspired spiking network modelled on the insect olfactory system. *Bioinspir. Biomim.* 11:026002. doi: 10.1088/1748-3190/11/2/026002

Diamond, A., Schmuker, M., and Nowotny, T. (2019). An unsupervised neuromorphic clustering algorithm. *Biol. Cybern.* doi: 10.1007/s00422-019-00797-7

Diehl, P. U., Pedroni, B. U., Cassidy, A., Merolla, P., Neftci, E., and Zarrella, G. (2016). TrueHappiness: neuromorphic emotion recognition on truenorth. *arXiv:1601.04183* [q-bio, cs]. Available online at: http://arxiv.org/abs/1601.04183 (accessed May 13, 2018).

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci.U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Gire, D. H., Franks, K. M., Zak, J. D., Tanaka, K. F., Whitesell, J. D., Mulligan, A. A., et al. (2012). Mitral cells in the olfactory bulb are mainly excited through a multistep signaling path. *J. Neurosci.* 32, 2964–2975. doi: 10.1523/JNEUROSCI.5580-11.2012

Gonzalez, J., Monroy, J. G., Garcia, F., and Blanco, J. L. (2011). "The multi-chamber electronic nose (MCE-nose)," in *2011 IEEE International Conference on Mechatronics* (Istanbul), 636−641.

Hige, T. (2018). What can tiny mushrooms in fruit flies tell us about learning and memory? *Neurosci. Res.* 129, 8–16. doi: 10.1016/j.neures.2017.05.002

Huerta, R., and Nowotny, T. (2009). Fast and robust learning by reinforcement signals: explorations in the insect brain. *Neural Comput.* 21, 2123–2151. doi: 10.1162/neco.2009.03-08-733

Imam, N., and Cleland, T. A. (2019). Rapid online learning and robust recall in a neuromorphic olfactory circuit. *arXiv:1906.07067 Cs Q-Bio*. Available online at: http://arxiv.org/abs/1906.07067 (accessed June 17, 2019).

Imam, N., Cleland, T. A., Manohar, R., Merolla, P. A., Arthur, J. V., Akopyan, F., et al. (2012). Implementation of olfactory bulb glomerular-layer computations in a digital neurosynaptic core. *Front. Neurosci.* 6:83. doi: 10.3389/fnins.2012.00083

Iskierko, Z., Sharma, P. S., Bartold, K., Pietrzyk-Le, A., Noworyta, K., and Kutner, W. (2016). Molecularly imprinted polymers for separating and sensing of macromolecular compounds and microorganisms. *Biotechnol. Adv.* 34, 30–46. doi: 10.1016/j.biotechadv.2015.12.002

Kashiwadani, H., Sasaki, Y. F., Uchida, N., and Mori, K. (1999). Synchronized oscillatory discharges of mitral/tufted cells with different molecular receptive ranges in the rabbit olfactory bulb. *J. Neurophysiol.* 82, 1786–1792. doi: 10.1152/jn.1999.82.4.1786

Kemker, R., and Kanan, C. (2017). FearNet: brain-inspired model for incremental learning. *arXiv:1711.10563* [cs]. Available online at: http://arxiv.org/abs/1711.10563 (accessed January 24, 2019).

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv:1412.6980* [cs]. Available online at: http://arxiv.org/abs/1412.6980 (accessed January 30, 2019).

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci.* 114, 3521–3526. doi: 10.1073/pnas.1611835114.

Länge, K., Rapp, B. E., and Rapp, M. (2008). Surface acoustic wave biosensors: a review. *Anal. Bioanal. Chem.* 391, 1509–1519. doi: 10.1007/s00216-008-1911-5

Lepousez, G., and Lledo, P.-M. (2013). Odor discrimination requires proper olfactory fast oscillations in awake mice. *Neuron* 80, 1010–1024. doi: 10.1016/j.neuron.2013.07.025

Lepousez, G., Nissant, A., Bryant, A. K., Gheusi, G., Greer, C. A., and Lledo, P.-M. (2014). Olfactory learning promotes input-specific synaptic plasticity in adult-born neurons. *Proc. Natl. Acad. Sci. U.S.A.* 111, 13984–13989. doi: 10.1073/pnas.1404991111

Li, G., and Cleland, T. A. (2017). A coupled-oscillator model of olfactory bulb gamma oscillations. *PLoS Comput. Biol.* 13:e1005760. doi: 10.1371/journal.pcbi.1005760

Linster, C., and Cleland, T. A. (2010). Decorrelation of odor representations via spike timing-dependent plasticity. *Front. Comput. Neurosci.* 4:157. doi: 10.3389/fncom.2010.00157

Liu, Q., Cai, H., Xu, Y., Li, Y., Li, R., and Wang, P. (2006). Olfactory cell-based biosensor: a first step towards a neurochip of bioelectronic nose. *Biosens. Bioelectron.* 22, 318–322. doi: 10.1016/j.bios.2006.01.016

Liu, Q., Hu, X., Ye, M., Cheng, X., and Li, F. (2015). Gas recognition under sensor drift by using deep learning. *Int. J. Intell. Syst.* 30, 907–922. doi: 10.1002/int.21731

Ma, Z., Luo, G., Qin, K., Wang, N., and Niu, W. (2018). Online sensor drift compensation for E-nose systems using domain adaptation and extreme learning machine. *Sensors* 18:E742. doi: 10.3390/s18030742

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Comput.* 8, 1–40. doi: 10.1162/neco.1996.8.1.1

Maass, W. (2015). To spike or not to spike: that is the question. *Proc. IEEE* 103, 2219–2224. doi: 10.1109/JPROC.2015.2496679

Mandairon, N., Kuczewski, N., Kermen, F., Forest, J., Midroit, M., Richard, M., et al. (2018). Opposite regulation of inhibition by adult-born granule cells during implicit versus explicit olfactory learning. *eLife* 7:e34976. doi: 10.7554/eLife.34976

McCloskey, M., and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: the sequential learning problem. *Psychol. Learn. Motiv.* 24, 109–165. doi: 10.1016/S0079-7421(08)60536-8

Mehta, D., Altan, E., Chandak, R., Raman, B., and Chakrabartty, S. (2017). "Behaving cyborg locusts for standoff chemical sensing," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD), 1–4.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Nagayama, S., Homma, R., and Imamura, F. (2014). Neuronal organization of olfactory bulb circuits. *Front. Neural Circuits* 8:98. doi: 10.3389/fncir.2014.00098

Nessler, B., Pfeiffer, M., and Maass, W. (2009). "STDP enables spiking neurons to detect hidden causes of their inputs," in *Advances in Neural Information Processing Systems 22*, eds Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta (Curran Associates, Inc.), 1357–1365. Available online at: http://papers.nips.cc/paper/3744-stdp-enables-spiking-neurons-to-detect-hidden-causes-of-their-inputs.pdf (accessed May 16, 2018).

O'Connor, P., Gavves, E., Reisser, M., and Welling, M. (2018). *Temporally Efficient Deep Learning with Spikes.* Available online at: https://openreview.net/forum?id=HyYuqoCUz (accessed May 18, 2018).

Panzeri, S., Brunel, N., Logothetis, N. K., and Kayser, C. (2010). Sensory neural codes using multiplexed temporal scales. *Trends Neurosci.* 33, 111–120. doi: 10.1016/j.tins.2009.12.001

Peace, S. T., Johnson, B. C., Li, G., Kaiser, M. E., Fukunaga, I., Schaefer, A. T., et al. (2017). Coherent olfactory bulb gamma oscillations arise from coupling independent columnar oscillators. *bioRxiv* 213827. doi: 10.1101/213827

Peng, F., and Chittka, L. (2017). A simple computational model of the bee mushroom body can explain seemingly complex forms of olfactory learning and memory. *Curr. Biol.* 27, 224–230. doi: 10.1016/j.cub.2016.10.054

Persaud, K., and Dodd, G. (1982). Analysis of discrimination mechanisms in the mammalian olfactory system using a model nose. *Nature* 299, 352–355. doi: 10.1038/299352a0

Persaud, K. C., Marco, S., and Gutiérrez-Gálvez, A. (eds.). (2013). *Neuromorphic Olfaction*. Boca Raton: CRC Press/Taylor & Francis.

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front. Neurosci.* 9:141. doi: 10.3389/fnins.2015.00141

Raman, B., and Gutierrez-Osuna, R. (2005). "Mixture segmentation and background suppression in chemosensor arrays with a model of olfactory bulb-cortex interaction," in *Proceedings 2005 IEEE International Joint Conference on Neural Networks* (Montreal, QC), Vol. 1, 131–136.

Raman, B., Sun, P. A., Gutierrez-Galvez, A., and Gutierrez-Osuna, R. (2006). Processing of chemical sensor arrays with a biologically inspired model of olfactory coding. *IEEE Trans. Neural Netw.* 17, 1015–1024. doi: 10.1109/TNN.2006.875975

Rodriguez-Lujan, I., Fonollosa, J., Vergara, A., Homer, M., and Huerta, R. (2014). On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemom. Intell. Lab. Syst.* 130, 123–134. doi: 10.1016/j.chemolab.2013.10.012

Schmiedt, J., Albers, C., and Pawelzik, K. (2010). "Spike timing-dependent plasticity as dynamic filter," in *Advances in Neural Information Processing Systems 23*, eds J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Curran Associates, Inc.), 2110–2118. Available online at: http://papers.nips.cc/paper/3917-spike-timing-dependent-plasticity-as-dynamic-filter.pdf (accessed May 18, 2018).

Schmuker, M., Nawrot, M., and Chicca, E. (2015). "Neuromorphic sensors, olfaction," in *Encyclopedia of Computational Neuroscience*, eds D. Jaeger and R. Jung (New York, NY: Springer), 1991–1997.

Schmuker, M., Pfeil, T., and Nawrot, M. P. (2014). A neuromorphic network for generic multivariate data classification. *Proc. Natl. Acad. Sci. U.S.A.* 111, 2081–2086. doi: 10.1073/pnas.1303053111

Serrà, J., Surís, D., Miron, M., and Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. *arXiv:1801.01423* [cs, stat]. Available online at: http://arxiv.org/abs/1801.01423 (accessed January 24, 2019).

Serrano, E., Nowotny, T., Levi, R., Smith, B. H., and Huerta, R. (2013). Gain control network conditions in early sensory coding. *PLoS Comput. Biol.* 9:e1003133. doi: 10.1371/journal.pcbi.1003133

Shi, H., Tsai, W.-B., Garrison, M. D., Ferrari, S., and Ratner, B. D. (1999). Template-imprinted nanostructured surfaces for protein recognition. *Nature* 398, 593–597. doi: 10.1038/19267

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Velez, R., and Clune, J. (2017). Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. *PLoS ONE* 12:e0187736. doi: 10.1371/journal.pone.0187736

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., and Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sens. Actuators B Chem.* 166–167, 320–329. doi: 10.1016/j.snb.2012.01.074

Xiong, W., and Chen, W. R. (2002). Dynamic gating of spike propagation in the mitral cell lateral dendrites. *Neuron* 34, 115–126. doi: 10.1016/S0896-6273(02)00628-1

Yan, K., Zhang, D., and Xu, Y. (2017). Correcting instrumental variation and time-varying drift using parallel and serial multitask learning. *IEEE Trans. Instrum. Meas.* 66, 2306–2316. doi: 10.1109/TIM.2017.2707898

Yin, H., Wang, Z., and Jha, N. (2018). A hierarchical inference model for internet-of-things. *IEEE Trans. Multi-Scale Comput. Syst.* 4, 260–271. doi: 10.1109/TMSCS.2018.2821154

Zaidi, Q., Victor, J., McDermott, J., Geffen, M., Bensmaia, S., and Cleland, T. A. (2013). Perceptual spaces: mathematical structures to neural mechanisms. *J. Neurosci.* 33, 17597–17602. doi: 10.1523/JNEUROSCI.3343-13.2013

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. *arXiv:1703.04200* [cs, q-bio, stat]. Available online at: http://arxiv.org/abs/1703.04200 (accessed January 24, 2019).

Zhang, L., and Zhang, D. (2015). Domain adaptation extreme learning machines for drift compensation in E-nose systems. *IEEE Trans. Instrum. Meas.* 64, 1790–1801. doi: 10.1109/TIM.2014.2367775

Zhang, Y., Zhao, J., Du, T., Zhu, Z., Zhang, J., and Liu, Q. (2017). A gas sensor array for the simultaneous detection of multiple VOCs. *Sci. Rep.* 7:1960. doi: 10.1038/s41598-017-02150-z

Conflict of Interest Statement: Both authors are listed as inventors on a Cornell University provisional patent (8631-01-US) covering other aspects of this algorithm.

ORIGINAL RESEARCH
published: 03 July 2019
doi: 10.3389/fnins.2019.00650

Constructing an Associative Memory System Using Spiking Neural Network

Hu He[1], Yingjie Shang[1], Xu Yang[2]*, Yingze Di[2], Jiajun Lin[2], Yimeng Zhu[2], Wenhao Zheng[2], Jinfeng Zhao[2], Mengyao Ji[2], Liya Dong[1], Ning Deng[1], Yunlin Lei[2] and Zenghao Chai[2]

[1] Institute of Microelectronics, Tsinghua University, Beijing, China, [2] School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China

Development of computer science has led to the blooming of artificial intelligence (AI), and neural networks are the core of AI research. Although mainstream neural networks have done well in the fields of image processing and speech recognition, they do not perform well in models aimed at understanding contextual information. In our opinion, the reason for this is that the essence of building a neural network through parameter training is to fit the data to the statistical law through parameter training. Since the neural network built using this approach does not possess memory ability, it cannot reflect the relationship between data with respect to the causality. Biological memory is fundamentally different from the current mainstream digital memory in terms of the storage method. The information stored in digital memory is converted to binary code and written in separate storage units. This physical isolation destroys the correlation of information. Therefore, the information stored in digital memory does not have the recall or association functions of biological memory which can present causality. In this paper, we present the results of our preliminary effort at constructing an associative memory system based on a spiking neural network. We broke the neural network building process into two phases: the Structure Formation Phase and the Parameter Training Phase. The Structure Formation Phase applies a learning method based on Hebb's rule to provoke neurons in the memory layer growing new synapses to connect to neighbor neurons as a response to the specific input spiking sequences fed to the neural network. The aim of this phase is to train the neural network to memorize the specific input spiking sequences. During the Parameter Training Phase, STDP and reinforcement learning are employed to optimize the weight of synapses and thus to find a way to let the neural network recall the memorized specific input spiking sequences. The results show that our memory neural network could memorize different targets and could recall the images it had memorized.

Keywords: spiking neural network, artificial intelligence, associative memory system, Hebb's rule, STDP

OPEN ACCESS

Edited by:
*Yansong Chua,
Institute for Infocomm Research
(A*STAR), Singapore*

Reviewed by:
*Alexantrou Serb,
University of Southampton,
United Kingdom
Arash Ahmadi,
University of Windsor, Canada*

***Correspondence:**
*Xu Yang
yangxu@tsinghua.edu.cn*

Specialty section:
*This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience*

Received: *28 January 2019*
Accepted: *06 June 2019*
Published: *03 July 2019*

Citation:
He H, Shang Y, Yang X, Di Y, Lin J, Zhu Y, Zheng W, Zhao J, Ji M, Dong L, Deng N, Lei Y and Chai Z (2019) Constructing an Associative Memory System Using Spiking Neural Network. Front. Neurosci. 13:650. doi: 10.3389/fnins.2019.00650

1. INTRODUCTION

Development of computer science has led to the blooming of artificial intelligence (AI). Research on AI has become extremely popular these days due to the ever-growing demands from application domains such as pattern recognition, image segmentation, intelligent video analytics, autonomous robotics, and sensorless control (Rowley et al., 1996; Lecun et al., 1998; Zaknich, 1998;

Egmont-Petersen et al., 2002). Neural networks are the core of AI research. Deep-learning neural networks (DNNs), the second generation of artificial neural networks (ANNs), have become the research hotspot of neural networks (Schmidhuber, 2014) and have won numerous contests against people, including the most famous one: recently, Google's AlphaGo DNN defeated Lee Sedol, a famous professional I-go player.

To date, many studies have been conducted on DNN, focusing on development of the learning and training methods (Jennings and Wooldridge, 2012; Yoshua et al., 2013; Lecun et al., 2015) of DNN. Researchers studying DNN typically use a fixed neural network structure and train their DNN using a large amount of data to optimize the weight of the connections/synapses.

Although the mainstream neural networks have done well in the fields of image processing and speech recognition, they do not perform well in models aimed at understanding contextual information. In our opinion, the reason for this is that the essence of building a neural network through parameter training is to fit the data to the statistical law through parameter training. Since the neural network built using this approach does not possess memory ability, it cannot reflect the relationship between data with respect to the causality. Recurrent neural networks (RNNs) use a special network structure to address this issue, but the complexity of its structure also leads to many limitations.

Spiking neural networks (SNNs) are the third generation of ANNs. Compared with DNNs, SNNs are more similar to the biological neural network; SNNs use spiking neurons, which emit spiking signals when activated. The generated spiking trains (sequences of spiking signals) are used to communicate between neurons. Spiking train expresses time dimension information naturally; therefore, SNNs offer an advantage when dealing with information having string contextual relevance. However, due to the lack of effective training algorithms, SNNs have not yet been applied to many domains. Many studies on SNNs have been published, but most of these involve using SNNs to perform simple classification or image recognition.

Neural networks in organisms can perform many complex functions, including memory. Since SNNs are more similar to the biological neural network, we endeavored to use it to construct a bionic memory neural network. Biological memory is fundamentally different from the current mainstream digital memory in terms of the storage method. The information stored in digital memory is converted to binary code and written in separate storage units. This physical isolation destroys the correlation of information. Therefore, the information stored in digital memory does not have the recall or association functions of biological memory which can present causality.

The great capability and potential of biological neural network fascinates us. So in this paper, we present our preliminary effort at constructing an associative memory neural network based on SNN. We present our method which could guide the grow process of the memory neural network. We present our method to optimize the weight of synapses of the neural network. And through our experimental results, we show that the memory neural network built using our method could possess memory and recall ability after only undergoing a small scale of training.

In our method, we broke the neural network building process into two phases: the Structure Formation Phase and the Parameter Training Phase. The Structure Formation Phase applies a learning method based on Hebb's rule to provoke neurons in the memory layer to new synapses to connect to neighbor neurons as a response to the specific input spiking sequences fed to the neural network. The aim of this phase is to train the neural network to memorize the specific input spiking sequences. During the Parameter Training Phase, STDP and reinforcement learning are employed to optimize the weight of synapses and thus find a way let the neural network recall the memorized specific input spiking sequences.

The remaining text is organized as follows: section 2 discusses related work, section 3 mentions our motivation, section 4 provides the study background, and section 5 discusses our method to implement the memory neural network; the experimental results are reported and discussed in section 6. The conclusion is provided in section 7.

2. RELATED WORKS

Neural network construction has a long history, and many algorithms have been proposed (Śmieja, 1993; Fiesler, 1994; Quinlan, 1998; Perez-Uribe, 1999).

As the second generation of ANNs, DNNs have many advantages. However, they rely heavily on data for training. With the construction of DNN becoming increasingly complex and powerful, the training process requires an increasing number of computations, which has become a great challenge. Each session of training becomes increasingly time and resource consuming, which may become a bottleneck for DNNs in the near future. Now, an increasing number of researchers are turning their attention to SNNs.

In 2002, Bohte et al. (2000) derived the first supervised training algorithm for SNNs, called SpikeProp, which is an adaptation of the gradient-descent-based error-back-propagation method. SpikeProp overcame the problems inherent to SNNs using a gradient-descent approach by allowing each neuron to fire only once (Wade et al., 2010). In 2010, Wade et al. presented a synaptic weight association training (SWAT) algorithm for spiking neural networks (SNNs), which merges the Bienenstock-Cooper-Munro (BCM) learning rule with spike timing dependent plasticity (STDP) (Wade et al., 2010).

In 2013, Kasabov et al. (2013) introduced a new model called deSNN, which utilizes rank-order learning and Spike Driven Synaptic Plasticity (SDSP) spike-time learning in unsupervised, supervised, or semi-supervised modes. In 2017, they presented a methodology for dynamic learning, visualization, and classification of functional magnetic resonance imaging (fMRI) as spatiotemporal brain data (Kasabov et al., 2016). The method they presented is based on an evolving spatiotemporal data machine of evolving spiking neural networks (SNNs) exemplified by the NeuCube architecture (Kasabov, 2014), which adopted both unsupervised learning and supervised learning in different phases.

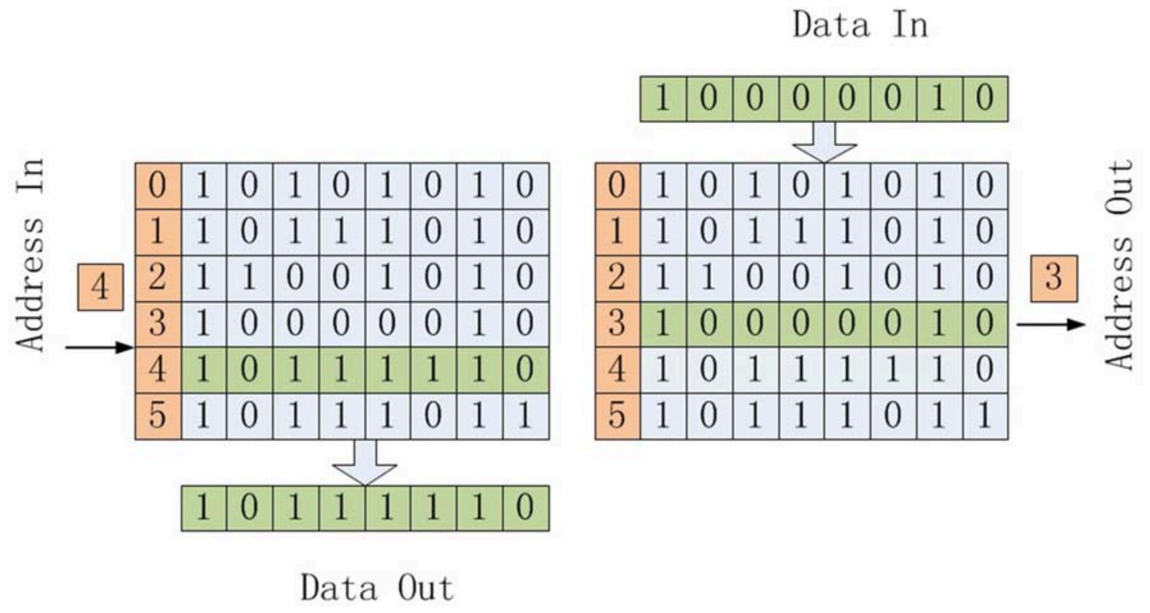


FIGURE 1 | Traditional Memory and CAM.

In 2019, He et al. (2019) proposed a bionic way to implement artificial neural networks through construction rather than training and learning. The hierarchy of the neural network is designed according to analysis of the required functionality, and then module design is carried out to form each hierarchy. The results show that the bionic artificial neural network built through their method could work as a bionic compound eye, which can achieve the detection of an object and its movement, and the results are better on some properties, compared with the Drosophila's biological compound eyes.

Some studies have already attempted to design neural networks that behave similar to a memory system. Lecun et al. (2015) proposed RNNs for time domain sequence data; RNNs use a special network structure to address the aforementioned issue, but the complexity of their structure also leads to many limitations.

Hochreiter and Schmidhuber (1997) presented the long short-term memory neural network, which is a variant of RNNs. This neural network inherits the excellent memory ability of RNNs with regard to the time series and overcomes the limitation of RNN, that is, difficulty in learning and preserving long-term information. Moreover, it has displayed remarkable performance in the fields of natural language processing and speech recognition. However, the efficiency and scalability of long short-term memory is poor.

Hopfield (1988) has established the Hopfield network, which is a recursive network computing model for simulating a biological neural system. The Hopfield network can simulate the memory and learning behavior of the brain. The successful application of this network to solve the traveling salesman problem shows the potential computing ability of the neural computing model for the NP class problem. However, the network capacity of the Hopfield network model is determined by neuron amounts and connections within a given network, thus the number of patterns that the network can remember is limited. Also, since patterns that the network uses for training (called retrieval states) become attractors of the system, repeated updates would eventually lead to convergence to one of the retrieval states. Thus, sometimes the network will converge to spurious patterns (different from the training patterns). And when the input patterns are similar, the network cannot always recall the correct memorized pattern, which means the fault-tolerance is affected by the relationship between input patterns.

3. MOTIVATION

In traditional memory, as shown in the left part of **Figure 1**, when we input an address, the memory outputs data stored in that address. In content addressable memory (CAM), as shown in the right part of **Figure 1**, when we input data, the address of that data is outputted.

In biological memory systems, both input and output are contents (**Figure 2**). Traditional memory and CAM can be cascaded to expand, as shown in **Figure 3**. However, due to the designing and addressing method of CAM, it is difficult to implement very large scale CAM. So, it is not able to implement cascaded CAM with large capacity in this way.

Biological memory systems are built on a neural network, which is composed of neurons. This kind of memory has a simple structure, large capacity, and can be easily expanded to a very large scale (**Figure 3**).

Therefore, the goal of this study was to build a bionic memory neural network.

4. BACKGROUND

4.1. Neuron Model

The leaky integrate and fire neuron model was used in this study (Indiveri, 2003). It is one of the most widely used models due to its computing efficiency. This model's behavior can be described as Equation 1.

$$V(t) = \begin{cases} \beta \cdot V(t-1) + V_{in}(t) & when\ V < V_{th} \\ V_{reset}\ and\ set\ a\ spike & when \quad V \geq V_{th} \end{cases}$$

where $V(t)$ is the state variable and β is the leaky parameter; V_{th} is the threshold state and V_{reset} is the reset state. Once $V(t)$ exceeds the threshold V_{th}, the neuron fires a spike and $V(t)$ is reset to V_{reset}.

4.2. Spiking Neural Networks

SNNs are inspired by the manner in which brain neurons function: through synaptic transmission of spiking trains. Spiking encoding integrates multiple aspects of information, such as time, space, frequency, and phase. It is an effective tool for complex space-time information processing. In addition, because SNNs contain time dimension information, its information processing ability is stronger than that of the previous two generations of neural networks, especially in the processing of information with strong contextual relevance.

There are many kinds of SNNs. In SNNs, all the information is encoded in spiking signals. Spiking trains, consisting of sequences of spiking signals, are transmitted in the neural network to implement communication between neurons.

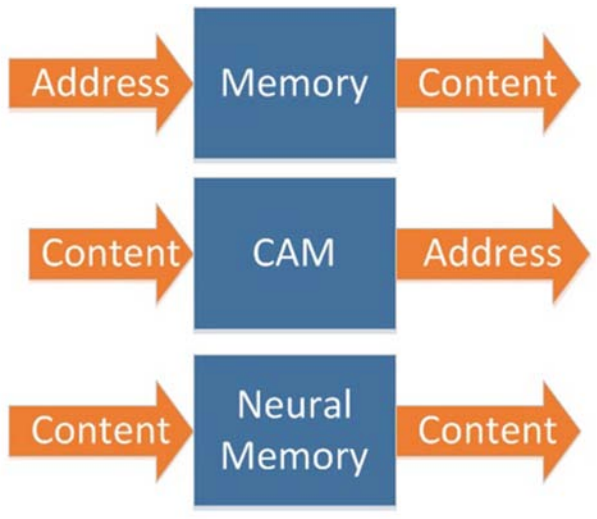


FIGURE 2 | Three different mechanism of memory.

4.3. Spike-Timing-Dependent Plasticity

Spike-timing-dependent plasticity (STDP) is one of the most important unsupervised learning rules in the SNNs. As a biological process, it describes the regulatory mechanism of synapses between neurons in the brain. In our method, STDP is used to guide the adjustment of the weight of synapses during the training of SNNs.

Let us suppose that there is a synapse from neuron N_{pre} to neuron N_{suc} in an SNN, and the firing time of N_{pre} is t_1 while that of N_{suc} is t_2. According to STDP, if $t_1 < t_2$, then the weight of the synapse from N_{pre} to N_{suc} should increase; if $t_1 > t_2$, then the weight of the synapse from N_{pre} to N_{suc} should decrease; if $t_1 = t_2$, then nothing should happen. The value of the increase/decrease in weights depends on the difference between t_1 and t_2.

4.4. Hebb's Learning Rule

The structure of a biological neural network is neither regular nor completely disordered, which is the result of the reflection to the input spiking sequences it receives. Or, we can say that it is the input spiking signals that define the structure of a biological neural network through learning and training. For example, in biological auditory systems, the structure of neural networks is related to their sensitivity to different frequencies of sound. However, the relationship between network structure and external stimulation is difficult to describe using a mathematical formula.

In our algorithm, we have applied a learning method based on Hebb's rule to form the structure of the memory neural network as a response or reflection of the input spiking sequences. Hebb's

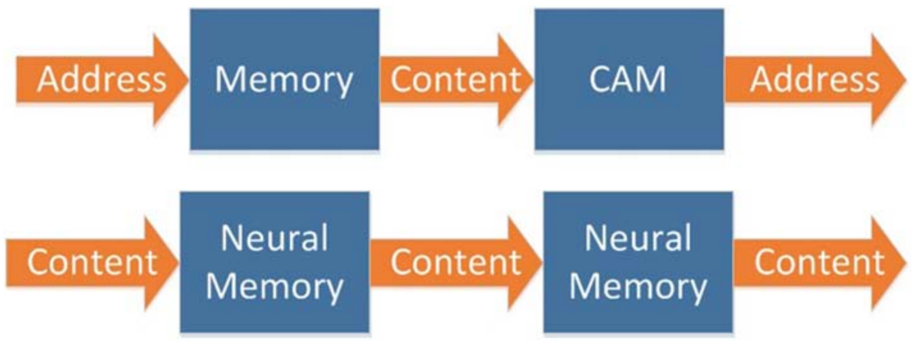


FIGURE 3 | Memory's cascading mechanism.

learning rule (Hebb, 1988) is a neuropsychological theory put forward by Donald Hebb in 1949. According to Hebb's learning rule (Hebb, 1988), when an axon of cell A is sufficiently close to excite a cell B, and repeatedly or persistently takes part in firing it, some growth-related process or metabolic changes take place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

5. METHOD TO CONSTRUCT BIONIC MEMORY NEURAL NETWORK

Our method for constructing the bionic memory neural network consists of four major phases:

1. Initialization phase: Initialize the input spiking sequences and initialize the neural network;
2. Structure Formation phase: Applies a learning method based on Hebb's rule to provoke neurons in the memory layer growing new synapses to connect to neighbor neurons as a response to the specific input spiking sequences fed to the input layer of the neural network, until the connection between the memory layer and the output layer is completed;
3. Parameter Training phase: STDP and reinforcement learning are employed to optimize and adjust the weight of synapses in the neural network;
4. Pruning phase: Comply with biological rules to delete unnecessary connections, thus enhancing the energy efficiency.

The detail process of our method is described in Algorithm 1.

In this work, the MNIST dataset (Lecun and Cortes, 2010) was selected to test our proposed method. The MNIST is a widely used dataset for optical character recognition, with 60,000 handwritten digits in the training set and 10,000 in the testing set. The size of handwritten digital images in this dataset is 28×28.

As stated in Algorithm 1, during the parameter training phase, we would test the memory neural network if it could recall the image it has already memorized. We would present one image from MNIST (already been processed and transferred into spiking sequence) to the input layer for a certain time duration. The input spiking sequence would be transferred to the memory layer. Neurons in the output layer would receive responses from the memory layer and fire if necessary, thus we could record the firing sequence from the output layer. Since one image would only be fed to the input layer for a limited time duration, after a while, there would be no more firing in the output layer, which indicates the end of the firing sequence. Then we will decide the meaning of this firing sequence by the majority votes method.

Algorithm 1: Experiment Process

```
Input:
    Input Image Set, $S$;
    Original Memory Neural Network, $NN$;
Output:
    Trained Memory Neural Network, $NN$;
1: Initialize the Input Spiking Sequences by employing the data preprocessing process to convert $S$ into spiking sequences set $SS$;
2: Initialize the memory neural network;
3: Set the turn mark of the Structure Formation phase, $TM_{SF} = 0$;
4: while $(TM_{SF} < 2)$ do
5:     Set the training set of the Structure Formation phase, $S_1 = SS$;
6:     while $(S_1 \neq \phi)$ do
7:         Pick one input spiking sequence $R$ from $S_1$, and delete it from $S_1$;
8:         Feed $R$ to the memory neural network, and perform Structure Formation phase;
9:     end while
10:    $TM_{SF} = TM_{SF} + 1$;
11: end while
12: Set the training set of the Parameter Training phase, $S_2 = SS$;
13: while $(S_2 \neq \phi)$ do
14:     Pick one input spiking sequence $R$ from $S_2$;
15:     Feed $R$ to the memory neural network;
16:     if Result of the output layer is correct then
17:         Delete $R$ from $S_2$;
18:     else
19:         Perform the Parameter Training phase for $R$;
20:     end if
21: end while
```

5.1. Initialization Phase

5.1.1. Initialize the Input Spiking Sequences

Since the input to our memory neural network should be spiking sequences, the MNIST images should first be transferred into the spiking sequence. When the input spiking sequences are initialized, a data preprocessing process is designed to convert the MNIST images into spiking sequences.

The data preprocessing process is shown in **Figure 4**.

The convolution layer and the pooling layer are added to abstract the features of the MNIST images, thus reducing the amount of information our memory neural network needs to memorize. Four 4×4 convolution kernels are used in the convolution layer, which are shown in **Figure 5**. MNIST images would be first processed by the four convolution kernels separately, then the result of the four convolution kernels would be processed by the pooling layer. The pooling layer employs 2×2 max_pooling operation.

The conversion layer is used to convert the images outputted by the pooling layer into spiking sequences according to the spiking encoding method. There are many kinds of encoding methods in literature. The principle of priority transmission of important information in the ROC (Rank Order Coding) coding method (Thorpe and Gautrais, 1998) is used to help design the encoding method in this paper. The spiking encoding method used in this paper converts the pixel value of the image into the delay time of the spiking signal, and the higher the pixel value is, the shorter the delay time is.

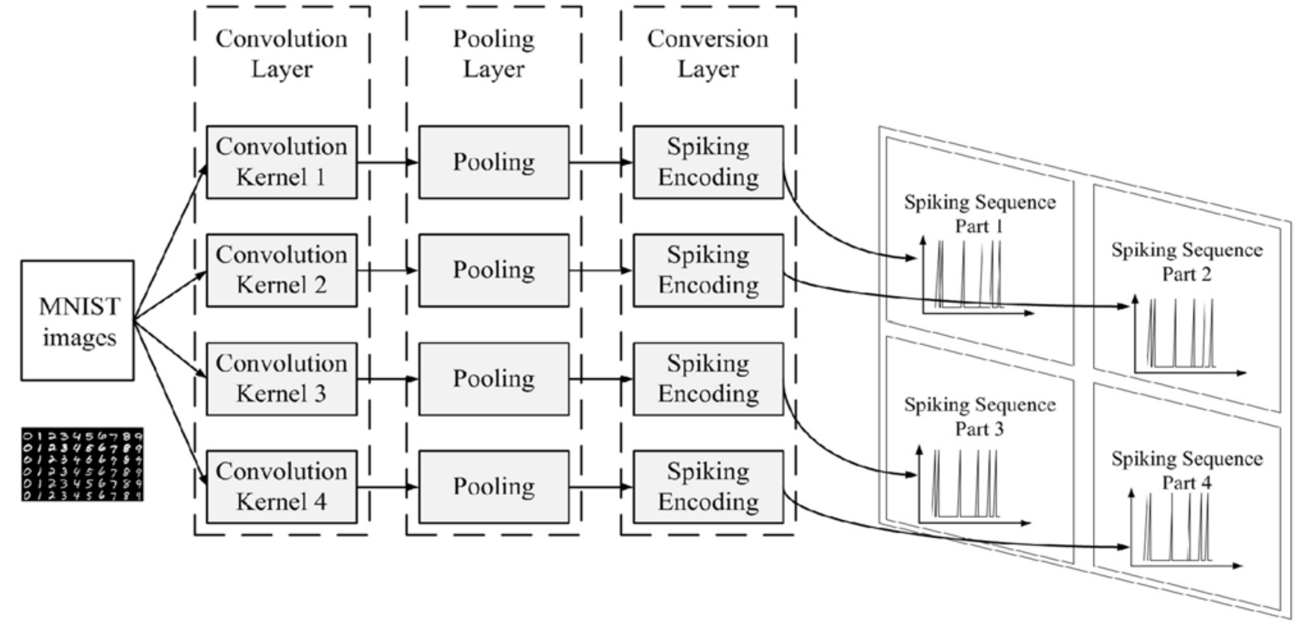


FIGURE 4 | Data preprocessing process for initializing MNIST images into input spiking sequences.

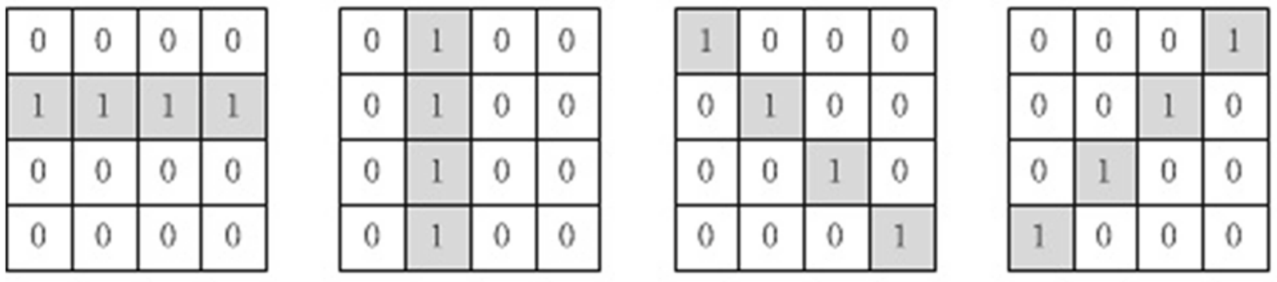


FIGURE 5 | Four convolution kernels used in our method.

Suppose the set of pixels in an image is D, then for each pixel $d \in D$, min_max normalization would first be employed to avoid the singular sample data affecting the convergence of the network:

$$R(d) = \frac{d - d_{min}}{d_{max} - d_{min}} \tag{1}$$

where d_{max} and d_{min} are the maximum and minimum value in D, respectively.

Four different spiking encoding methods have been designed in this paper:

Method 1: Linear encoding method, where $S(d) = T_{max} - R(d) \times (T_{max} - T_{min})$;
Method 2: Exponential encoding method, where $S(d) = (0.5^{R(d)-1} - 1) \times (T_{max} - T_{min}) + T_{min}$;
Method 3: Inverse encoding method, where $S(d) = (\frac{2}{R(d)+1} - 1) \times (T_{max} - T_{min}) + T_{min}$;
Method 4: Power encoding method, where $S(d) = (R(d) - 1)^2 \times (T_{max} - T_{min}) + T_{min}$.

where T_{max} and T_{min} are the stop time and start time of the spiking sequence for that image, while $S(d)$ is the converted spiking time for pixel d.

The relationship between the pixel value and the spiking time for those four methods is compared in **Figure 6**. In the graph, the horizontal coordinates represent the pixel values, while the vertical coordinates are the encoded spiking times. According to the comparison, we can conclude that the power encoding method could emit more important information in an earlier time, thus we chose the power encoding method as the spiking encoding method for this paper.

The pixel value range of the MNIST images is [0,255]. After being processed by the conversion layer, an image from the MNIST set would be converted into an input spiking sequence with spiking signals in a time range of [0, 100 ms].

5.1.2. Initialize the Neural Network

Our memory neural network consists of three layers: the input layer, the memory layer and the output layer, as shown in **Figure 7**. The input layer is in charge of receiving input spiking sequences and feeding the input spiking sequences into the memory layer. The memory layer would grow new connections as a response of input spiking sequences to remember them, then through proper training recall them and output the correct result through the output layer. The output layer exists because we not only want our neural network to possess memory ability, but also to be able to output recall result. The number of neurons in the output layer is set as the same as the number of targets that the memory neural network needs to be memorized.

The task of this initialization phase is to initialize all three layers and initialize the connections between the input layer and the memory layer. The number of neurons in the input layer is determined by the size of the target to be memorized. As shown in **Figure 9**, neurons in the input layer are connected to neurons in the memory layer with a one-to-one style. So, the number of neurons in the memory layer is same as the input layer. The weight of synapses in this work is set in the range [0, 100]. In order to provoke enough responses in the memory layer to allow the learning method based on Hebb's rule to work, the initialized weight of connection from the input layer to the memory layer should be strong enough, and is set as 50 in this work.

Since the original MNIST image is 28×28, after the operation of the four convolution kernels in the convolution layer, the result is 4 parts each with sizes of 25×25, and after the pooling layer, the result is 4 parts each with sizes of 12×12. Since the result after the pooling layer is 4 parts each with sizes of 12×12, there are 576 spiking signals in the spiking sequence in total after the process of the conversion layer. Thus, in this work, we set 576 neurons in the input layer of our memory neural network. Each spiking signal in the spiking sequence would feed into one of the input neurons. And since the connection style between input layer and memory layer is one-to-one, there are also 576 neurons in the memory layer in this work.

Ten images, each of different number (that is one image of each from 0 to 9), are chosen from MNIST to form the Input Image Set *S* of this work, as shown in **Figure 8**. Thus the number of neurons in the output layer is 10, corresponding to the 10 images needed to be memorized.

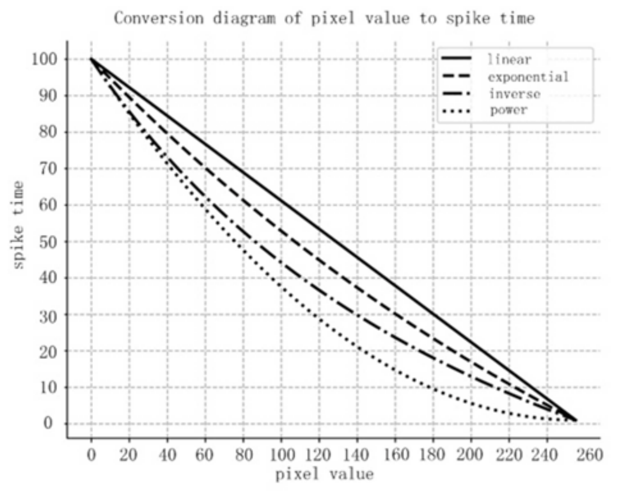


FIGURE 6 | Comparison of the four encoding methods.

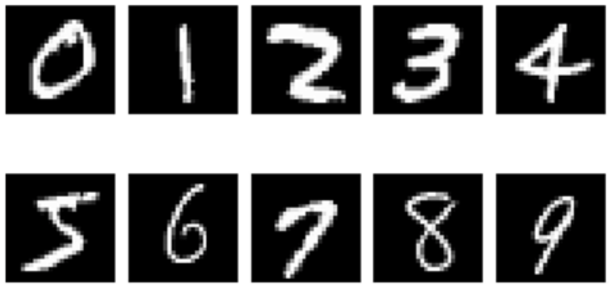


FIGURE 8 | Input Image Set *S*.

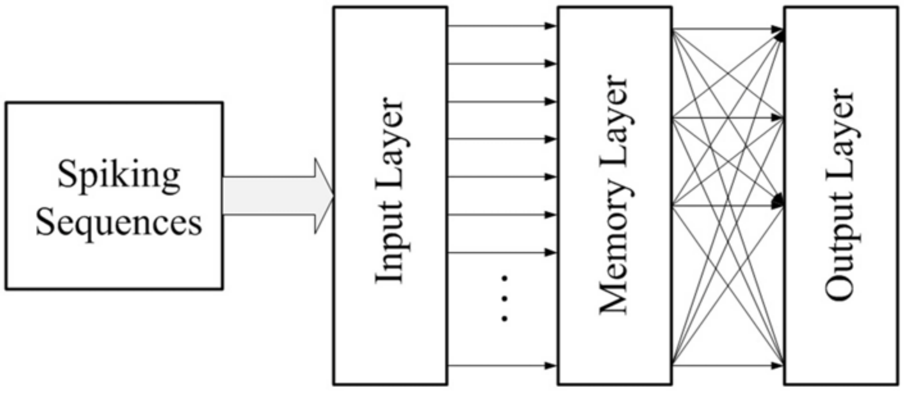


FIGURE 7 | Structure of our memory neural network.

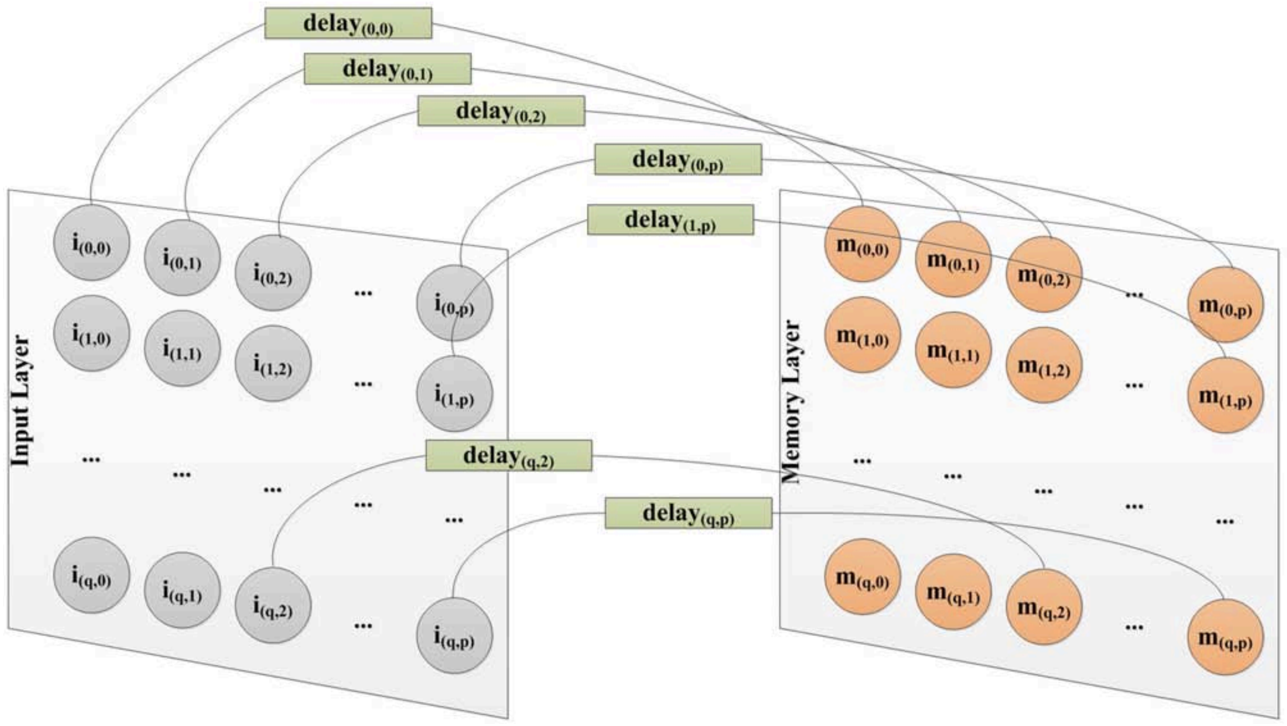

delay(0,0)
delay(0,1)
delay(0,2)
delay(0,p)
delay(1,p)
delay(q,2)
delay(q,p)
Input Layer
Memory Layer
i(0,0) i(0,1) i(0,2) i(0,p)
i(1,0) i(1,1) i(1,2) i(1,p)
i(q,0) i(q,1) i(q,2) i(q,p)
m(0,0) m(0,1) m(0,2) m(0,p)
m(1,0) m(1,1) m(1,2) m(1,p)
m(q,0) m(q,1) m(q,2) m(q,p)

FIGURE 9 | Different delay for each connection from input layer to memory layer to capture spatial information.

Each neuron in the input layer and memory layer will be assigned a coordinate, as shown in **Figure 9**. The coordinates of neurons in the memory layer would be used to calculate the distance between them in later course of our algorithm.

As stated before in the paper, we use MNIST images as the input. There are two kinds of information in a Mnist image. The value of the pixel, and the location of that pixel. We use the power encoding method to convert the value of the pixel into the spiking time of that pixel. And in order to capture the spatial information of the pixels, we have implemented a spatial-to-temporal mechanism to decide the delay of a connection from neurons in the input layer to neurons in the memory layer, as shown in **Figure 9**. The delay of a connection from neuron $i_{(x,y)}$ in a $p \times q$ input layer to neuron $m_{(x,y)}$ in a $p \times q$ memory layer is calculated as:

$$delay_{im(x,y)} = x * p + y + 1 \tag{2}$$

here (x, y) is the coordinate of that neuron.

This acts as a way to encode spatial information into temporal information, which then could be captured by SNNs.

5.2. Structure Formation Phase

During the structure formation phase, input spiking sequences would be fed to the input layer of the memory neural network, which would then be fed to the memory layer through connections between the input layer and the memory layer. The behavior of all the neurons in the memory layer would be recorded. Additionally, a learning method is conducted to direct the growing of new connections in the memory layer.

According to Hebb's learning rule (Hebb, 1988), when an axon of cell A is sufficiently near to excite a cell B, and repeatedly or persistently takes part in firing it, some growth-related process or metabolic changes take place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

A learning method based on Hebb's learning rule is designed to direct the growing of new connections (synapses) in the structure formation phase. According to our learning algorithm, if the firing times of two neurons are very close, and there is no connection between them, a connection is established between them. In order to prevent the explosive growth of network connections, our approach considers the coordinate of neurons and does not establish connections when the Euclidean distance between neurons exceeds a pre-defined threshold.

The detail description of this algorithm is provided below:

Step 1: Start the simulation, record firing behaviors of neurons in the memory layer;
Step 2: Examine whether there exists a pair of neurons N_1 and N_2 in the memory layer such that both have fired during the simulation, and the distance between neurons N_1 and N_2 satisfies that $Dis(N_1\ to\ N_2) < Dis_{threshold}$ (where $Dis_{threshold}$ is a pre-defined distance threshold for our algorithm). If any, proceed to Step 3; otherwise, proceed to Step 4;
Step 3: Suppose the firing time of N_1 is t_1, and that of N_2 is t_2. If $0 < abs(t_1 - t_2) < Threshold$ and $(t_1 < t_2)$, establish a connection from N_1 to N_2 with weight of 10, and proceed to

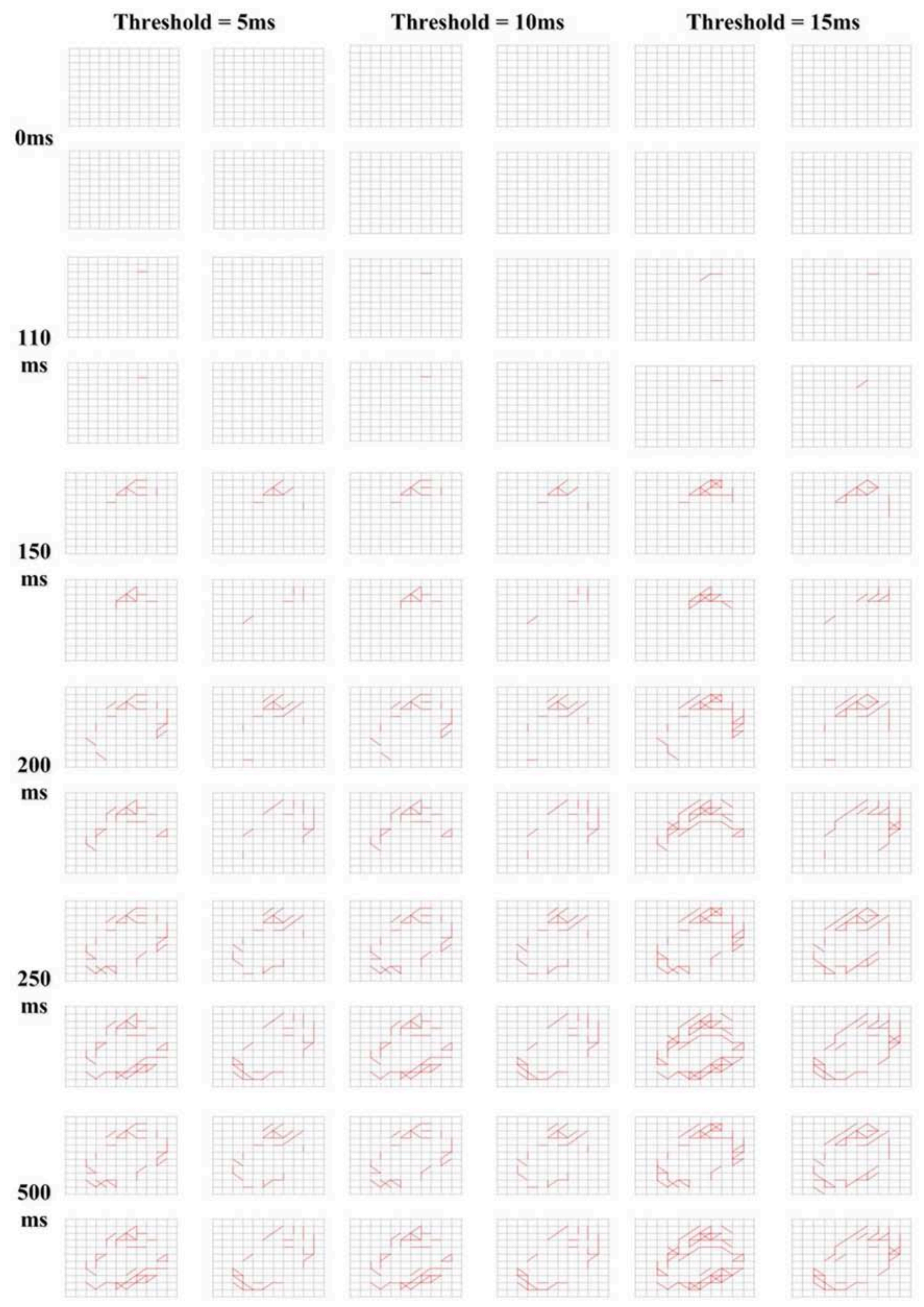


FIGURE 10 | Different growing behavior due to different learning *Threshold*.

Step 4; if $0 < abs(t_1 - t_2) < Threshold$ and $(t_1 > t_2)$, establish a connection from N_2 to N_1 with weight of 10, proceed to Step 4; if $abs(t_1 - t_2) \geq Threshold$, proceed to Step 4;
Step 4: If the stop criterion is satisfied, end the simulation; otherwise, go to Step 2.

Since the connections in the memory layer are grown under guidance of the learning method based on Hebb's learning rule, the distance threshold $Dis_{threshold}$ is used to control the number of connections generated in the memory layer. If the threshold is smaller, then there would be less connections. If the threshold is larger, there would be more connections. The $Dis_{threshold}$ in this work is set as 2.

This process continues until the stop criterion is satisfied. Then, neurons in the memory layer are connected to the neurons in the output layer according to their firing behavior. As we

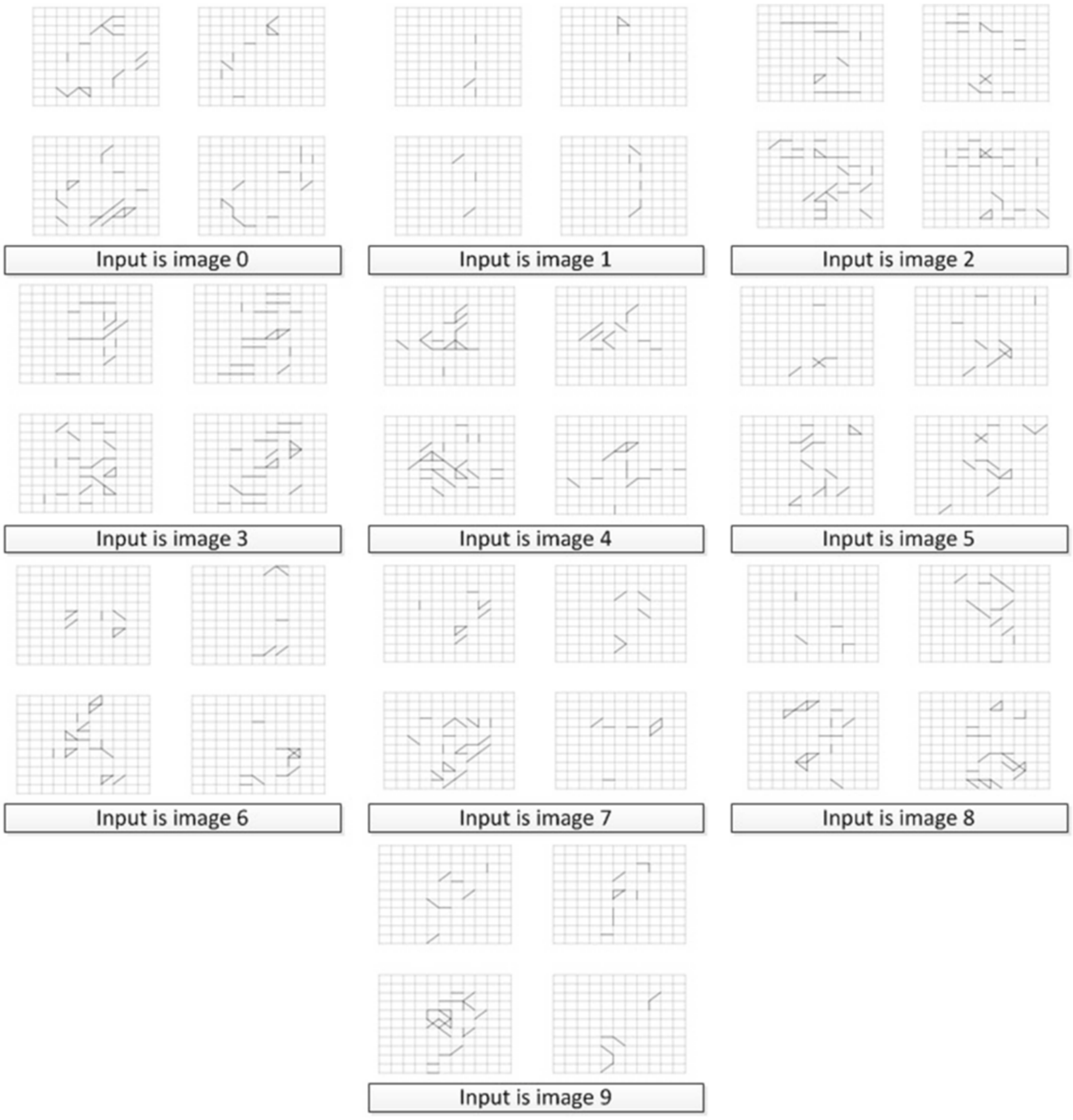


FIGURE 11 | Different memory layer structure to memory different input images.

have discussed in section 5.1, a spatial-to-temporal mechanism has been introduced to decide the delay of a connection from input layer to memory layer, since the neuron model we used is a LIF model. In order to avoid the unnecessary reduction of firing activity of neurons in the output layer, due to the leaking characteristics of the LIF model, we have also implemented a temporal-to-spatial mechanism to calculate the delay of connection from neurons in the memory layer to neurons in the output layer. The delay of a connection formed between neuron $m_{(x,y)}$ in the memory layer and neuron o_z in the output layer is calculated as:

$$delay_{mo(x,y)} = [N_m - delay_{im(x,y)}] + 1 \tag{3}$$

where N_m is the total number of neurons in the memory layer, while (x, y) is the coordinate of neurons in the memory layer as shown in **Figure 9**.

In our opinion, if a neuron in the memory layer fired when we fed the input spiking sequence related to a specific target, then it has causality with the memory behavior of that specific target. Since neurons in the output layer correspond to the targets needed to be memorized, we connect neurons in the memory layer which fired when we fed the input spiking sequence related to a specific target, to the neuron in the output layer which represents that specific target. The initialized weight of a connection established this way is $weight/n$, where $weight$ is a pre-defined constant, and n is the number of neurons in the memory layer which are connected to that neuron in the output layer. This is an approximate process. The weight of connections from neurons in the memory layer to neruons in the memory layer or connections from neurons in the memory layer to neurons in the output layer would be optimized during the parameter training phase.

5.3. Parameter Training Phase

Through structure formation phase, we have made the neural network to memorize specific targets represented by input spiking sequences. However, as a memory, we still need to have a recall mechanism. When fed the specific input spiking sequence again, which the neural network has already memorized, the memory neural network needs to recall it and output a correct result, represented by the correct behavior of the output layer. During the parameter training phase, we will rely on STDP and reinforcement learning to optimize the weight of connections (synapses) in the neural network to implement the recall mechanism. The weight of connections between the input layer and memory layer would not be optimized during this phase. In the parameter training phase, the STDP option of NEST (the evaluation platform we used for this work) is always on.

The algorithm for parameter training phase is described below:

Step 1: Pick one input from the input spiking sequences training set;
Step 2: Feed the picked input to the input layer and examine the result sequence of the output layer;
Step 3: If the result sequence of the output layer is correct, go to Step 1; Otherwise go to Step 4;

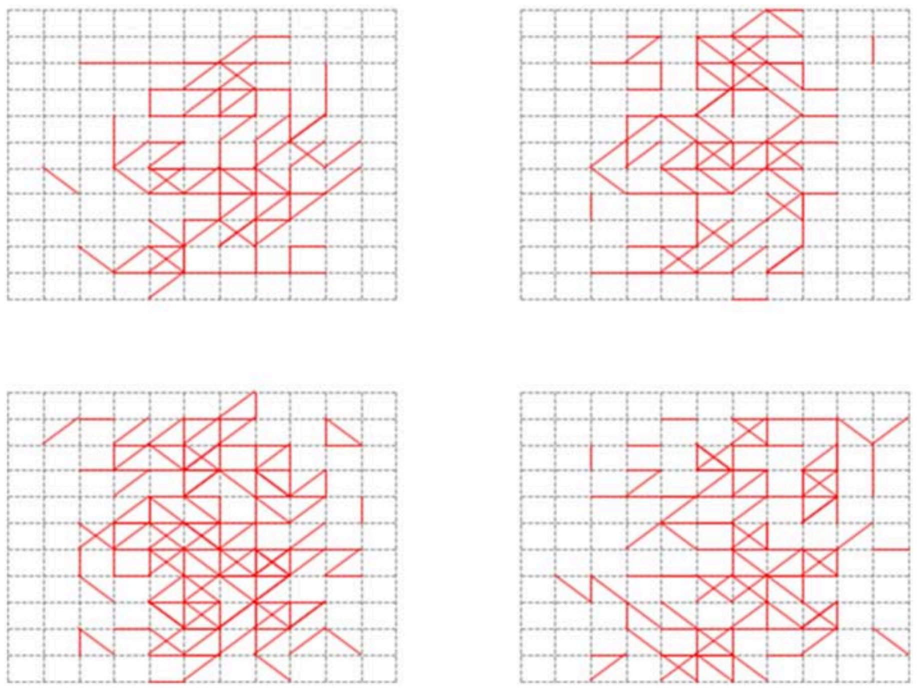


FIGURE 12 | Generated memory neural network with learning *Threshold* of 5 ms.

Step 4: Identify the set of incorrectly firing neurons in the output layer as S_O and identify the set of firing neurons in the memory layer as S_M;

Step 5: If neuron i is in S_M, and neuron j is in S_O, and there is a connection from neuron i to neuron j, suppose the weight of this connection is $W_{i,j}$, then $W_{i,j} = W_{i,j} * Shrink\_Coeff$, and go to Step 2;

During the parameter training phase, when a specific input spiking sequence is fed to the input layer to train the memory neural network, the firing behavior of the neurons in the output

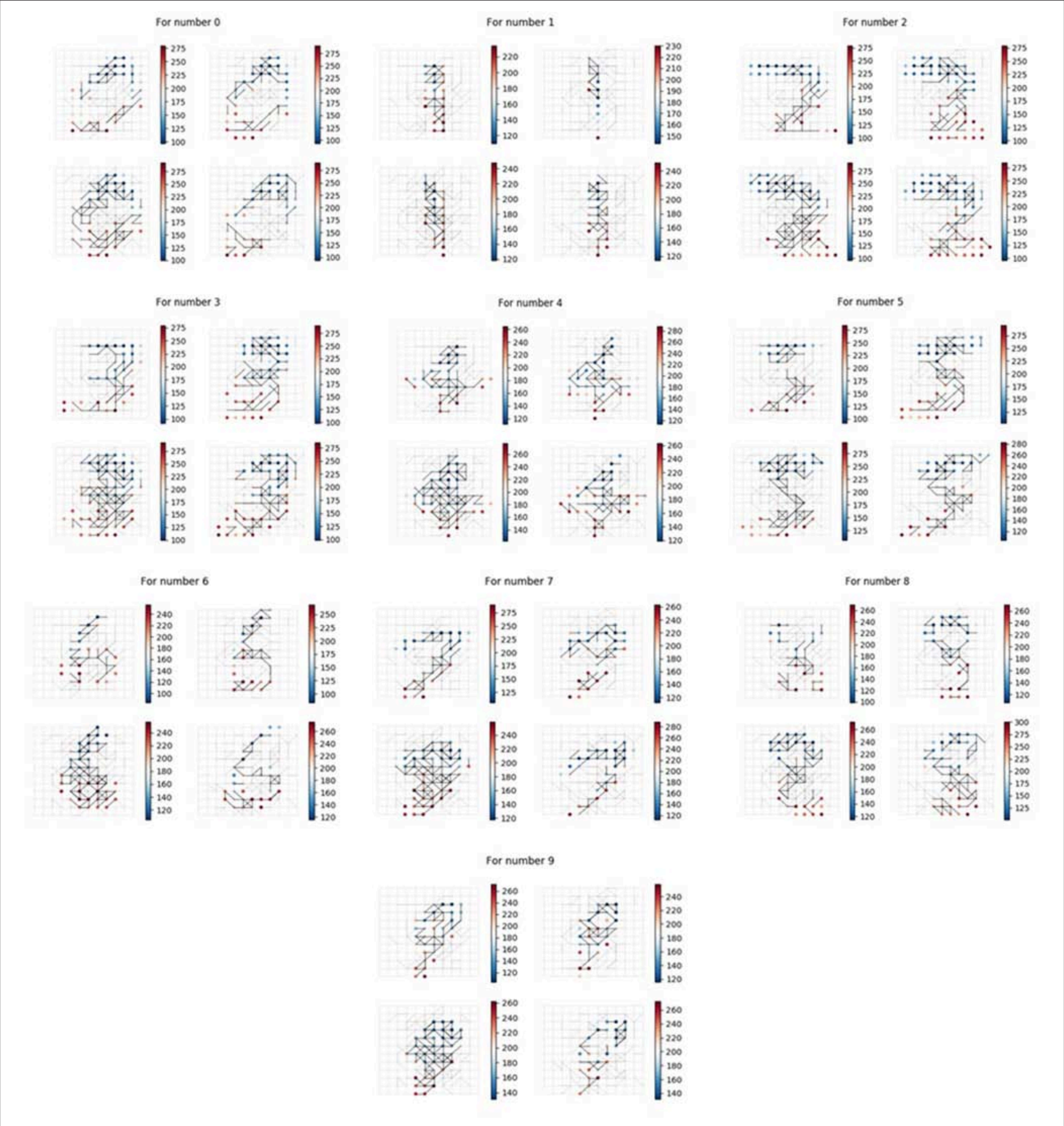


FIGURE 13 | Recall response for images in the Input Image Set.

layer would be recorded. The label corresponding to the most frequently fired neuron in the output layer is identified as the output result for this specific input spiking sequence. If the result is correct, then we suppose the memory neural network could correctly recall. If not, optimization needs to be done to establish the right recall mechanism.

TABLE 1 | Recall test result for memory neural network.

Label	Firing sequence of output neurons	Output	Result
0	[9 0 7 4 6 0 0 3 7 6 5 0]	0	Correct
1	[1]	1	Correct
2	[9 7 2 5 3 4 0 8 1 6 2 7 8 9 5 2 2]	2	Correct
3	[9 8 5 3 6 7 9 0 2 8 3 5 7 9 3 3]	3	Correct
4	[1 9 4 9 4 6 4 4]	4	Correct
5	[5 7 9 0 6 3 5 5]	5	Correct
6	[6 9 6 4 5 8 6 6]	6	Correct
7	[9 7 9 7 7 9 7]	7	Correct
8	[8 8 5 2 6 3 9 7 4 8 8]	8	Correct
9	[9 7 9 9 6 7 4 9]	9	Correct

As we said before, causality is the basis on which we built our method. If a specific input spiking sequence is fed to the input layer of the memory neural network, but the most frequently fired neuron in the output layer is not the correct one, it means that some of the fired neurons in the memory layer have contributed to the result under incorrect causality and thus need to be corrected while the contribution needs to be weakened.

The algorithm would seek out those connections, and STDP and reinforcement-based methods are used to optimize the weight of those connections, as shown in the algorithm description.

5.4. Pruning Phase

One of the most important advantages of the biological neural network is its energy efficiency. In our method, we introduced the pruning phase to delete redundant and unnecessary connections from the trained neural network. The method examines the weight of all connections. If the weight of a connection is smaller than a pre-defined threshold (set as 3 in this work), that connection is deleted. Further, if a neuron has no output connection, all the input connections of that neuron are also

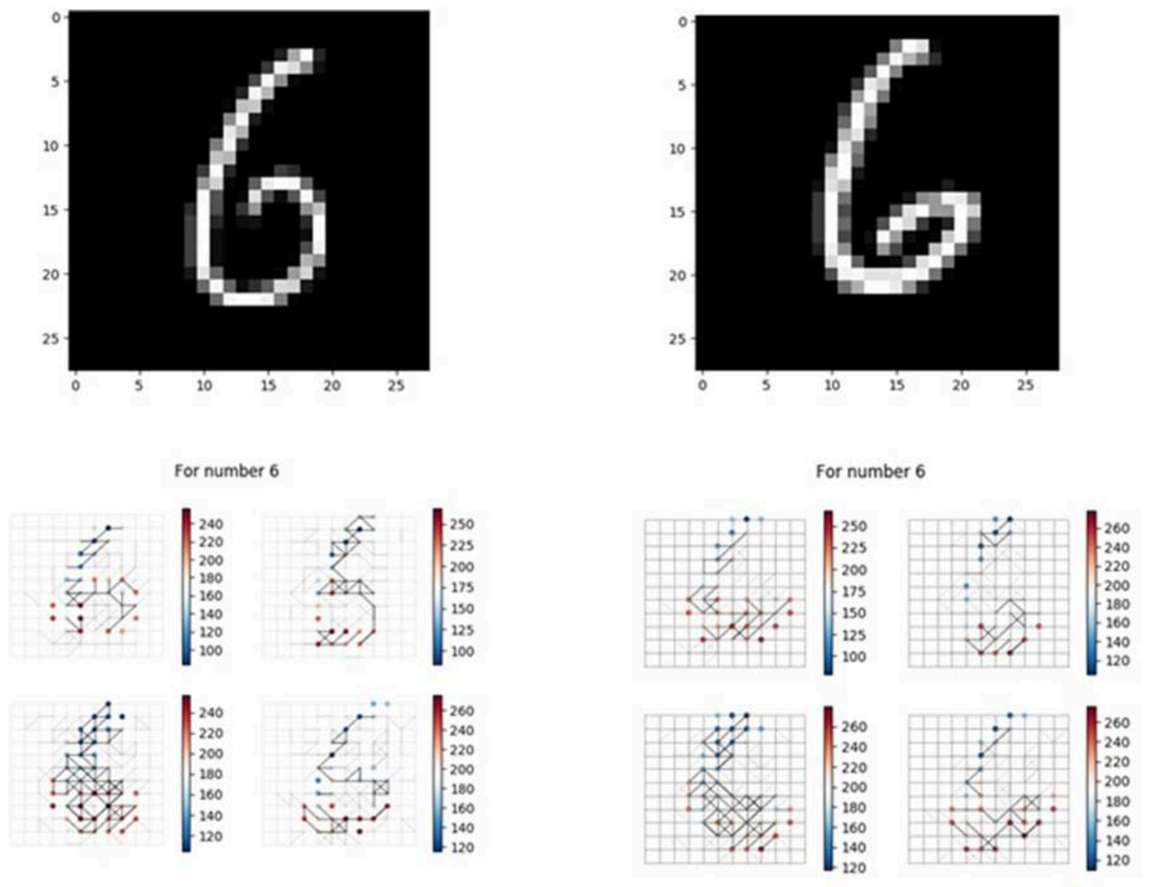


FIGURE 14 | Verification of the association ability.

deleted. The pruning phase helps enhance the energy efficiency of the neural network.

6. EXPERIMENT RESULTS

6.1. Evaluation Framework

We built our simulation platform based on the neural simulation tool NEST (Plesser et al., 2015), which is a simulation platform specially designed for SNN research. Biological spiking neural networks are characterized by the parallel operation of thousands of spiking neurons and the exchange of information between them by spiking trains sent via synapses. This mode of functioning fits the characteristics of the message passing interface parallel mechanism in particular. NEST supports message passing interface parallelization. Further, NEST provides users a method of asynchronous multi-process concurrent execution, which makes the program execute the model asynchronously and efficiently, and automatically synchronizes the process during the simulation without user interaction. Parallel computing reduces the time required and increases the scale of operations.

We conducted two sets of experiments. In the first set of experiments, in order to show the difference between the structures of the memory layer when used to memorize different targets, we used 10 identical SNNs to train 10 different images each, numbered from “0” to “9.” In the second set of experiments, we used 1 SNN to train on all those 10 images to test the recall (with those 10 images it already memorized) and association (using an image it has not seen before) ability.

6.2. Results and Discussion

6.2.1. Growing Process of the Memory Layer

After the Initialization phase, there was no connection in the memory layer. During the Structure Formation phase, when the input spiking sequences are fed to the input layer of our memory neural network, under the control of the learning method, new connections would grow in the memory layer. An illustration of the growing process of the memory layer during Structure Transformation phase under different *Threshold* value choices is shown in **Figure 10**. The 4x different subpanels in each relevant panel correspond to the parts in the memory layer which are the output of each kernel. The input image is a “0” from the MNIST set. From the comparison we could conclude that, when the *Threshold* is smaller, the connection in the memory layer is more sparse, thus the memory layer could remember more due to the larger available capacity.

6.2.2. Results of Memory Process

In order to verify that our memory neural network could remember different targets, we conducted the first set of experiments and built 10 memory neural networks, each fed with a different image numbered from 0 to 9 (as shown in **Figure 8**). The results of the memory layer after the Structure Formation phase are shown in **Figure 11**, and the learning *Threshold* was set to 5 ms. The 4x different subpanels in each relevant panel correspond to the parts in the memory layer which are the output of each kernel. Each memory neural network is trained with only 1 image. According to **Figure 11**, we could see that our memory neural network could grow different connections in the memory layer to memory different targets.

6.2.3. Results of Recall Process

In order to test the recall ability of our memory neural network, we conducted the second set of experiment. First, we used all the images in the Input Image Set *S* as shown in **Figure 8** to perform the Structure Formation phase. Then we used the images in the Input Image Set *S* again to perform the Parameter Training phase and the Pruning phase. The memory layer of the generated memory neural network is shown in **Figure 12**. The 4x different subpanels in each relevant panel correspond to the parts in the memory layer which are the output of each kernel.

Figure 13 shows the firing behavior of the memory layer when we feed the images from the Input Image Set *S* to the generated memory neural network. The 4x different subpanels in each relevant panel correspond to the parts in the memory layer which are the output of each kernel. Different color represents different firing time, as shown in the vertical coordinate line beside each sub-figure. It could be seen that different images would provoke different parts in the memory layer to respond and generate different firing behavior. As described in section 5, when an image is fed to the memory neural network, a firing sequence of output neurons would be observed to decide the output result for that image using the majority votes method. The results are recorded in **Table 1**.

The results show that our memory neural network could recall the images it has memorized.

6.2.4. Verification of the Association Ability

We also want to test whether, if we feed images that our memory neural network has not seen before but are similar with the images it has memorized, it has the association ability to give a correct result. **Figure 14** shows one of the example tests. The 4x different subpanels in each relevant panel correspond to the parts in the memory layer which are the output of each kernel. The memory neural network used is the one generated in the second set of experiments. The left top part is the image used in the process to generate our memory neural network, while the right top image is a new one to test the association ability.

The left bottom part is the recall response of the left top image, while the right bottom part is the response of the memory layer when the new one is fed to the memory neural network. When the left top image is fed to the memory neural network, the firing sequence observed in the output layer is [6 9 6 4 5 8 6 6], and when the right top image is fed to the memory neural network, the firing sequence observed in the output layer is [6 9 4 6]. So when fed with unseen (unmemorized) but similar images, our memory neural network could illustrate some degree of association ability.

7. CONCLUSION

In this paper, we presented our effort at constructing an associative memory neural network through SNNs. We broke the neural network building process into two phases: the Structure Formation Phase and the Parameter Training Phase. The Structure Formation Phase applies a learning method based on Hebb’s rule to provoke neurons in the memory layer growing new synapses to connect to neighbor neurons as a response to the specific input spiking sequences fed to the neural network. The aim of this phase is to train the neural network to memorize the specific input spiking sequences. During the Parameter Training Phase, STDP and reinforcement learning are employed to optimize the weight of synapses, to find a way to allow the neural network to recall the memorized specific input spiking sequences.

Results show that, when the input spiking sequences are fed to the input layer of our memory neural network, under the control of the learning method, new connections would grow in the memory layer, and learning the *Threshold* value could be used to control the sparsity of the generated memory layer. Experiments show that our memory neural network was able to memorize different targets and could recall the images it has memorized. Further experimentation showed that when fed with unseen (unmemorized) but similar images, our memory neural network could also illustrate some degree of association ability.

Future work might include: (1)To teach our memory neural network to memorize more complex targets; (2) to enhance our memory neural network's association ability; (3) to grow our memory neural network into a large-scale memory inference system using our method; and (4) the goal of constructing a memory system with causality reasoning nearly the size of a biological brain.

DATA AVAILABILITY

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

YD, JL, and YZ were in charge of data curation. WZ and JZ were in charge of formal analysis. HH, YS, and XY were in charge of methodology. MJ and LD were in charge of software. YL was in charge of validation. ZC was in charge of visualization. XY was in charge of writing. HH, ND, and XY were in charge of funding acquisition.

FUNDING

This work was supported by the National Natural Science Foundation of China (under Grant No. 91846303), National Natural Science Foundation of China (under Grant No. 61502032), and Tsinghua and Samsung Joint Laboratory.

REFERENCES

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: a review and new perspectives. *IEEE Trans. Patt. Analy. Mach. Intell.* 35, 1798–1828. doi: 10.1109/TPAMI.2013.50

Bohte, S. M., La Poutre, J. A., and Kok, J. N. (2000). *Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons*. Technical Report, CWI (Centre for Mathematics and Computer Science), Amsterdam.

Egmont-Petersen, M., Ridder, D. D., and Handels, H. (2002). Image processing with neural networks: a review. *Pattern Recogn.* 35, 2279–2301. doi: 10.1016/S0031-3203(01)00178-9

Fiesler, E. (1994). "Comparative bibliography of ontogenic neural networks," in *International Conference on Artificial Neural Networks* (Sorrento), 26–29.

He, H., Yang, X., Xu, Z., Deng, N., Shang, Y., Liu, G., et al. (2019). Implementing artificial neural networks through bionic construction. *PLoS ONE* 14:e0212368. doi: 10.1371/journal.pone.0212368

Hebb, D. O. (1988). "The organization of behavior," in *Neurocomputing: Foundations of Research*.

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neuralcomputation* 9, 1735–1780.

Hopfield, J. J. (1988). Artificial neural networks. *IEEE Circ. Dev. Magaz.* 4, 2–10. doi: 10.1109/101.8118

Indiveri, G. (2003). "A low-power adaptive integrate-and-fire neuron circuit," in *International Symposium on Circuits and Systems* (Bangkok). doi: 10.1109/ISCAS.2003.1206342

Jennings, N. R., and Wooldridge, M. J. (2012). *Foundations of Machine Learning* (MIT Press).

Kasabov, N., Dhoble, K., Nuntalid, N., and Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition. *Neural Netw.* 41, 188–201. doi: 10.1016/j.neunet.2012.11.014

Kasabov, N. K. (2014). Neucube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Netw.* 52, 62–76. doi: 10.1016/j.neunet.2014.01.006

Kasabov, N. K., Doborjeh, M. G., and Doborjeh, Z. G. (2016). Mapping, learning, visualization, classification, and understanding of fmri data in the neucube evolving spatiotemporal data machine of spiking neural networks. *Trans. Neural Netw. Learn. Syst.* 99, 1–13. doi: 10.1109/TNNLS.2016.2612890

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521:436–444. doi: 10.1038/nature14539

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324.

Lecun, Y., and Cortes, C. (2010). *The Mnist Database of Handwritten Digits.* Available online at: http://yann.lecun.com/exdb/mnist/

Perez-Uribe, A. (1999). *Structure-Adaptable Digital Neural Networks*.

Plesser, H. E., Diesmann, M., Gewaltig, M. O., and Morrison, A. (2015). *NEST: the Neural Simulation Tool.* New York, NY: Springer.

Quinlan, P. T. (1998). Structural change and development in real and artificial neural networks. *Neural Netw. Off. J. Int. Neural Net. Soc.* 11, 577–599.

Rowley, H. A., Baluja, S., and Kanade, T. (1996). Neural network-based face detection. *IEEE Trans. Patt. Analy. Mach. Intell.* 20, 203–208.

Schmidhuber, J. (2014). Deep learning in neural networks: an overview. *Neural Netw.* 61, 85–117. doi: 10.1016/j.neunet.2014.09.003

Śmieja, F. J. (1993). Neural network constructive algorithms: trading generalization for learning efficiency? *Circ. Syst. Signal Proc.* 12, 331–374.

Thorpe. S., and Gautrais, J. (1998). "Rank order coding," in *Computational Neuroscience* (Springer).

Wade, J. J., Mcdaid, L. J., Santos, J. A., and Sayers, H. M. (2010). Swat: a spiking neural network training algorithm for classification problems. *IEEE Trans. Neural Netw.* 21, 1817–1830. doi: 10.1109/TNN.2010.2074212

Zaknich, A. (1998). Introduction to the modified probabilistic neural network for general signal processing applications. *IEEE Trans. Signal Proc.* 46, 1980–1990.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 04 July 2019
doi: 10.3389/fnins.2019.00686

Deep Liquid State Machines With Neural Plasticity for Video Activity Recognition

Nicholas Soures* and Dhireesha Kudithipudi

Neuromorphic AI Laboratory, Rochester Institute of Technology, Rochester, NY, United States

Real-world applications such as first-person video activity recognition require intelligent edge devices. However, size, weight, and power constraints of the embedded platforms cannot support resource intensive state-of-the-art algorithms. Machine learning lite algorithms, such as reservoir computing, with shallow 3-layer networks are computationally frugal as only the output layer is trained. By reducing network depth and plasticity, reservoir computing minimizes computational power and complexity, making the algorithms optimal for edge devices. However, as a trade-off for their frugal nature, reservoir computing sacrifices computational power compared to state-of-the-art methods. A good compromise between reservoir computing and fully supervised networks are the proposed deep-LSM networks. The deep-LSM is a deep spiking neural network which captures dynamic information over multiple time-scales with a combination of randomly connected layers and unsupervised layers. The deep-LSM processes the captured dynamic information through an attention modulated readout layer to perform classification. We demonstrate that the deep-LSM achieves an average of 84.78% accuracy on the DogCentric video activity recognition task, beating state-of-the-art. The deep-LSM also shows up to 91.13% memory savings and up to 91.55% reduction in synaptic operations when compared to similar recurrent neural network models. Based on these results we claim that the deep-LSM is capable of overcoming limitations of traditional reservoir computing, while maintaining the low computational cost associated with reservoir computing.

Keywords: spiking, LSM, local learning, deep, recurrent

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Hesham Mostafa,
University of California, San Diego,
United States
Arindam Basu,
Nanyang Technological University,
Singapore

****Correspondence:***
Nicholas Soures
nms9121@rit.edu

Specialty section:
This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: *01 March 2019*
Accepted: *17 June 2019*
Published: *04 July 2019*

Citation:
Soures N and Kudithipudi D (2019)
Deep Liquid State Machines With
Neural Plasticity for Video Activity
Recognition. Front. Neurosci. 13:686.
doi: 10.3389/fnins.2019.00686

1. INTRODUCTION

Enabling intelligence on the edge minimizes the round trip delay in decision-making, lowers communication costs, load-balances for the end user, and enhances security with caching or local algorithms to pre-process the data. An emerging input source for edge devices is streaming visual data from first person cameras, such as in smart vehicles, or wearable devices. Being able to accurately process streaming video is crucial for edge devices to understand and react to their environment in a wide range of applications (*eg: path planning, action selection, or surveillance*). A popular application for demonstrating understanding of first-person video data in machine learning and computer vision is video activity recognition. However, majority of state-of-the-art methods for video activity recognition do not target low-end embedded platforms. Complex networks are not amenable for on-device intelligence due to their compute and memory intensive operations (networks with 10–60 million synapses require 0.32–2 GB to store synaptic weights Alom et al., 2018) and long training times (in the order of hours to days with GPUs Fu and Carter, 2016).

In the early 2000s, a computationally light algorithm known as reservoir computing (RC) was proposed by two research groups independently. The two algorithms are otherwise known as the Echo State Network (ESN) (Jaeger, 2001) and the Liquid State Machine (LSM) (Maass et al., 2002). The main difference between the two is that the LSM is a biologically inspired spiking neural network (SNN), whereas the ESN is a rate-based approximation. In this work we focus on the LSM, a neurally inspired algorithm, with innate characteristics for edge devices that bring in size, weight, and power constraints. In particular SNNs can store the neuronal activation's in a single bit (all or nothing signal), can consume as low as $\approx$ $20pJ$ per spike (Neftci et al., 2017), and shown to be computationally at least as powerful as sigmoid and threshold neurons (Maass, 1997).

The LSM is a three-layer neural network which consists of an input layer, a liquid layer, and a readout layer. The recurrent connections in the liquid layer allow it to capture dynamic information, where information fades out over time. The advantage of the LSM is that all the synaptic connections, except for those which connect to the readout layer, are randomly initialized and remain fixed. Unique inputs will produce distinct perturbations in the state of the high-dimensional liquid layer from which information can be extracted. By using fixed connections, the LSM can circumvent the need for expensive learning rules and the problem of vanishing gradients which can impede learning with gradient descent approaches in recurrent neural networks. In Soures et al. (2017), it was shown that these networks are robust to internal noise, making them a natural choice for embedded systems, particularly analog implementations which are prone to device noise. However, the conventional LSM model has shown limited applicability in complex real-world problems owing to the single dynamical layer driven by an input signal (Hermans and Schrauwen, 2013; Ma et al., 2017). The single layer constricts the temporal dynamics of the LSM resulting in very large reservoir networks to solve trivial tasks. Another drawback with LSM is its dependence on the initialization of random synaptic connections. Recent literature highlights the gaps in conventional LSM, RC networks in general, and the need to extend the capabilities of these networks (Jaeger, 2007; Triefenbach et al., 2010, 2013; Gallicchio and Micheli, 2016; Wang and Li, 2016; Ma et al., 2017; Bellec et al., 2018). Motivated by these observations, we propose a novel framework that drastically reduces the overall computational resources without sacrificing the overall performance in complex spatiotemporal task. Specific contributions of this work are

1. Deep-LSM, a semi-trained deep spiking recurrent neural network with LSM as a core building block, capable of capturing information over multiple time-scales.
2. Demonstrate that a modular/deep architecture significantly reduces the memory requirements for storing synaptic weights.
3. Use local, unsupervised plasticity mechanisms to partially train the network yields state-of-the-art performance while minimizing the cost of training.
4. Design an attention modulated readout layer to selectively process information in the deep-LSM with limited computational resources.
5. Analyze the model performance on first-person video activity recognition with DogCentric dataset (Iwashita et al., 2014) and demonstrate state-of-the-art performance.
6. Observe $\approx$ 90% memory savings and reduction in number of operations compared to a LSTM and $\approx$ 25% reduction of memory consumption in comparison to a standard LSM and 16% decrease in number of operations.

2. RELATED WORK

2.1. Video Activity Recognition

Egocentric video activity recognition is quickly becoming a pertinent application area due to first person wearable devices such as body cameras or in robotics. In these application domains, real-time learning is critical for deployment beyond controlled environments (such as deep space exploration), or to learn continuously in novel scenarios. Many research groups have focused on solving video activity recognition problems with 2D and 3D convolutions (Tran et al., 2015), optical flow (Simonyan and Zisserman, 2014; Zhan et al., 2014; Ma et al., 2016; Song et al., 2016a), hand-crafted features (Ryoo et al., 2015), combining motion sensors with visual information (Song et al., 2016a,b), or using long-short term memory (LSTM) networks to capture dynamics about spatial information extracted by a convolutional neural network (CNN) (Baccouche et al., 2011; Yue-Hei Ng et al., 2015). These approaches, while befitting for high-end compute platforms, are often not suitable for wearable devices due to the resource intensive networks or the long training times.

Efficient video activity recognition designed for mobile devices has been studied by several research groups. An energy aware training algorithm was proposed in Possas et al. (2018), to demonstrate energy efficient video activity recognition on complex problems. In this work, the authors use reinforcement learning to train a network on both video and motion information captured by sensors while penalizing actions that have high energy costs. Another approach to minimizing energy consumption in mobile devices when using an accelerometer for activity recognition is to minimize the sampling rate (Zheng et al., 2017). In Yan et al. (2012) and Lee and Kim (2016), the authors investigate a network with adaptive features, sampling frequency, and window size for minimizing energy consumption during activity recognition.

Recently Graham et al. (2017) proposed convolutional drift networks (CDNs) for enabling real-time learning on mobile devices. CDNs are an architecture for video activity recognition which use a pre-trained CNN to extract features from video frames and an ESN to capture temporal information. The motivation behind the CDNs is to minimize the training time and compute resources for spatiotemporal tasks when compared to networks akin to LSTMs (Yue-Hei Ng et al., 2015; Graham et al., 2017). A similar sized RC network requires one fourth of the weights, has faster training, and lower energy consumption as that of an LSTM.

2.2. Hierarchical Reservoir Computing

As conventional reservoir networks are shallow and capture information in short time-scales, recently several research groups have investigated hierarchical reservoir models. A hierarchical ESN is introduced in Jaeger (2007) with the goal of developing a hierarchical information processing system which feeds on high-dimension time series data and learns its own features and concepts with minimal supervision. The hierarchical layers help the system to process information on multiple timescales where faster information is processed in the earlier layers and information on slower timescales is processed in the final layers. The outputs of each reservoir feed sequentially into the next reservoir in the network. The networks prediction is made from a combination of all the reservoir outputs. More recently, a hierarchical ESN was proposed in Ma et al. (2017). In this work the authors explore the use of trained auto-encoders, principal component analysis, and random connections as encoding layers between each reservoir layer. The downside to this approach is that the output layer is trained on the activity of every encoding layer, the last reservoir, and the current input. This means as the number of layers increases, the output layer size will increase. Another hierarchical model was developed in Triefenbach et al. (2010). This model is implemented by stacking trained ESNs on top of each other to create a hierarchical chain of reservoirs. The hierarchical ESN is applied to speech recognition where the intermediary layers have a readout layer trained to perform the tasks and the inputs to the hierarchical layers are the predictions of the previous layers. With this approach each layer corrects the error from the previous layer. The authors later designed a hierarchical ESN where each layer was trained on a broad representation of the output, which became more specific at later layers (Triefenbach et al., 2013). Another hierarchical ESN proposed in Gallicchio and Micheli (2016) connects an ensemble of ESNs together. In Carmichael et al. (2018), our group has proposed a mod-deepESN architecture, a modular architecture that allows for varying topologies of deep ESNs. Intrinsic plasticity mechanism is embedded in the ESN that contributes more equally toward predictions and achieves better performance with increased breadth and depth. In Wang and Li (2016), a deep LSM model is proposed for image processing which uses multiple LSMs as filters with a single response. The authors use convolution and pooling similar to the process of CNNs and train the LSMs with an unsupervised learning rule. In Bellec et al. (2018), the authors introduce an approximation of backpropagation-through-time for LSMs to optimize the temporal memory of the LSM. The network shows a large improvement in performance on sequential MNIST and speech recognition with the TIMIT speech corpus. Another approach to optimizing the LSM is Roy and Basu (2016), which proposes a computationally efficient on-line learning rule for unsupervised optimization of reservoir connections.

This work aims to develop an algorithm that overcomes few of the gaps in the vanilla RC network while focusing on maintaining the inherent efficiency of LSMs.

3. DEEP-LSM MODEL

The proposed deep-LSM, shown in **Figure 1**, is a network comprised of deep randomly initialized hidden layers to capture the key dynamics of input streams. Sandwiched between the hidden layers, unsupervised winner-take-all (WTA) layers encode a low-dimensional representation of the dynamic information captured by the high-dimensional hidden layer. The encoded representation is then passed to the next hidden layer in the network. The main role of the WTA layer is to extract features from the hidden layer to represent its dynamic behavior as a low dimensional input. As data flows through the deep-LSM, different hidden layers process information over multiple time-scales. The main elements of the proposed deep-LSM are optimization of short-term plasticity and initialization of the random hidden layers, the use of spike-timing dependent plasticity (STDP) to implement the unsupervised WTA layers, and the attention modulated readout layer.

3.1. Hidden Layer Optimization

The hidden layers in the deep-LSM are similar to the liquid layer in the LSM. The connections between neurons in the input layer to the hidden layer are random and sparse. The probability of a connection is drawn from a uniform random distribution and the degree of sparsity varies based on the application and number of input signals. In Litwin-Kumar et al. (2017), the authors state that the granule cells produce a 10–30x increase in dimensionality. They also highlight that the granule cells need to connect to a sparse number of inputs to produce a unique high-dimensional representation. Using these claims as guiding principles for the initialization of the hidden layer, the number of neurons is set to be approximately 10x the size of the input space in this work. The hidden layer consists of two populations of neurons, primary neurons which are connected to the input layer, and auxiliary neurons which only have recurrent connections within the hidden layer but do not connect to the input layer. Each primary neuron only connects to a sparse number of input neurons, creating a selective response such that no neuron responds to the same feature or set of features. The auxiliary neurons then help to capture dynamic information through their recurrent connections and propagate information through the network.

The hidden layer in this work is implemented with excitatory (E) and inhibitory (I) leaky integrate-and-fire neurons whose dynamics are modeled by (1).

$$\tau \frac{\partial V}{\partial t} = -V + I_{ext} * R, \qquad (1)$$

When a neuron recieves a pre-synaptic spike, the current is modeled by a square pulse of current with a magnitude proportional to the synaptic strength for 3 ms after the spike occurs. The LIF neurons are instantiated as a 3D grid of neurons

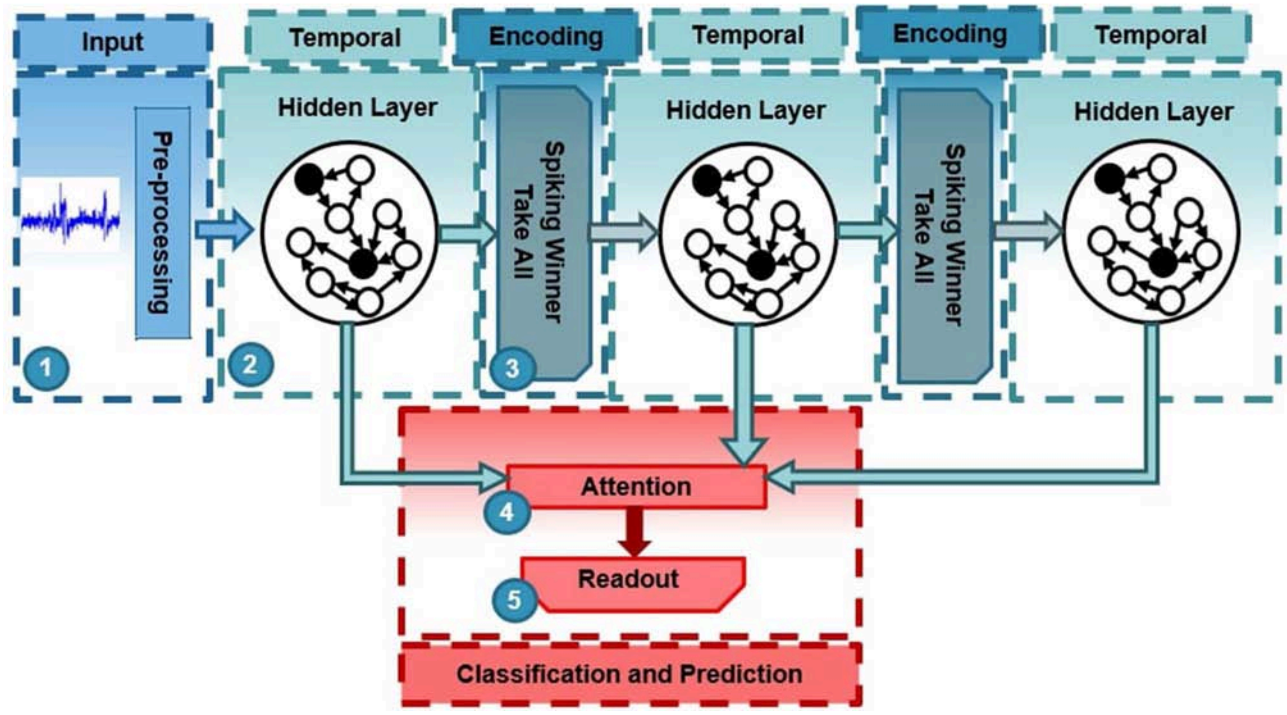


FIGURE 1 | Architecture of deep-LSM with three layers. (1) The input signals are randomly projected to a high dimensional space in the first hidden layer. (2) Hidden layers with random recurrent connections capture temporal information. (3) Spiking winner-take-all layers extract temporal features from the random hidden layers through the deep-LSM. (4) An attention function condensed the representation of the deep-LSM's hidden layers for efficient classification. (5) A readout layer is trained to perform a specific task.

with a ratio of 4:1 for the number of excitatory to inhibitory neurons. The probability of a recurrent connection forming is computed by (2).

$$\Pr\left(w_{i,j}^{res} \neq 0\right) = C\exp\left(-D(i,j)/\lambda\right)^2, \tag{2}$$

Where the probability of a connection depends on a scalar C (determined by the neuron types and the direction of the connection) which sets the maximum probability of a connection, and the Euclidean distance between the neurons scaled by λ which controls how quickly the probability of a connection drops off as the distance increases. The recurrent connections are initialized using fixed weights for each connection type where excitatory to excitatory (EE) connections have a synaptic strength of 3, EI have a strength of 3, IE have a strength of 4, and II have a strength of 1. In Renart et al. (2003) it was shown that neurons having homogeneous excitability is important in the dynamics of temporal memory. To maintain a homogeneous excitability in the hidden layer, the excitatory and inhibitory pre-synaptic connections are normalized so the sum of excitatory synapses and sum of inhibitory synapses is consistent for all neurons.

Another biologically inspired mechanism in the hidden layer is the use of short-term plasticity (STP). STP acts as a form of hidden memory in the hidden layer by reflecting a neurons recent firing activity. It also helps to regulate the overall firing activity by reducing the strength of spikes from highly active neurons. To optimize the STP function for neuromorphic systems, we reduce the computational cost of the STP equations from Markram et al. (1998) to (3) which simplified the model from an exponential function to a simple linear model.

$$S(n) = S(n-1) - \alpha * (x(n) - \beta) \tag{3}$$

where S is the synaptic efficacy regulating the strength of a neurons action potential and is bounded between 0 and 1. If a neuron emits a spike ($x(n) = 1$), the strength of S is decreased and if $x(n) = 0$ then S is increased. α and β are hyper-parameters used to control the dynamics of STP. A timestep of 1ms is used for all results presented in this work. The benefits of the STP rule in 3 are (i) changes in synaptic efficacy are constant and, (ii) are not dependent on the previous state of the synaptic efficacy.

The outputs of the hidden layer need to be sent to a readout layer to perform classification or prediction. If a binary state matrix (i.e., if a neuron fired) is used to represent the hidden layer's activity, several states collapse upon each other which can impact the networks ability to distinguish the different temporal patterns. Typically an exponential filtering operation is performed on the output of each neuron in the hidden layer (Schrauwen et al., 2007). In this work a synaptic trace operation is implemented at the output of each hidden neuron before transmitting to the readout layer which does not require the computation of any exponential terms. This operation is given by Equation (4)

$$\frac{dX^{trace}}{dn} = \frac{-X^{trace}}{\tau_{trace}} + \sum_{n^f} \delta(n - n^f) \tag{4}$$

where the synaptic trace (X_{trace}) keeps track of the behavior of the spike activity of a neuron (x(n)) by increasing the trace by a count of one every time a spike occurs and slowly decaying over time. This trace value is used by the readout layer to perform classification and prediction by capturing the short term behavior of each hidden neuron.

3.2. Deep-LSM Implementation

In Jaeger (2007), the authors provide evidence that deep networks are computationally more efficient and powerful than a shallow (single-layer) architecture. A deep model allows the network to learn more complex abstractions of the input and process the input on different timescales in the case of RNNs (Jaeger, 2007). Therefore the deep-LSM can extract higher level temporal features in each subsequent hidden layer before finally sending the information to a readout layer.

The inputs to each layer in the deep-LSM can be described by Equations (5)–(7)

$$I_{L_1}(n) = W^{in}_{L_1} * u(n) + W^{rec}_{L_1} * x_{L_1}(n-1) \tag{5}$$

$$I_{E_k}(n) = W^{in}_{E_k} * x_{L_{l=k}}(n) \tag{6}$$

$$I_{L_l}(n) = W^{in}_{L_l} * x_{E_{k=l-1}}(n) + W^{rec}_{L_l} * x_{L_l}(n-1) \tag{7}$$

where (5) is the input to the first hidden layer L_1 which combines information from the input layer $u(n)$ and input from the spiking activity of the hidden layer x_{L_1} through the recurrent connections. The input to the k^{th} WTA layer is described by (6) where $x_{L_{l=k}}(n)$ is the spiking activity of the previous hidden layer. Lastly, (7) is the input to the l^{th} hidden layer which receives the spiking activity at the current timestep from the previous encoding layer $x_{E_{k=l-1}}(n)$ and input about the hidden layer's previous spiking activity $x_{L_l}(n-1)$ through recurrent connections. In this architecture there is always one more hidden layer than the number of WTA layers because the activity of the hidden layer is what is used for classification.

In the deep-LSM architecture shown in **Figure 2**, the synaptic connections from the input layer to the first hidden layer, and from the WTA layers to the hidden layers are sparse. The synaptic connections from the hidden layers to the WTA layers (represented by dashed lines) are fully connected and trained with Spike-time Dependent Plasticity (STDP). STDP is a form of hebbian learning which postulates that neurons which fire together grow together (Hebb, 1949). In this case if a pre-synaptic potential occurs before a post-synaptic potential the synaptic strength is increased and vice-versa, if a post-synaptic potential occurs before a pre-synaptic potential the synaptic strength is decreased.

A simple learning rule based on a pre-synaptic trace from Diehl and Cook (2015) is used to model STDP. The pre-synaptic trace is a function which tracks the recent activity of the pre-synaptic neurons given by (4). The unsupervised learning rule can then be defined as

$$\Delta W_{i,j} = \alpha * (X^{trace}_j - X^{tar}) \tag{8}$$

where α is a hyper-parameter to control the magnitude of the weight change. The change in the synaptic strength between pre-synaptic neuron j and post-synaptic neuron i is increased proportional to the difference between the trace of pre-synaptic activity X^{trace}_j and the threshold activity level X^{tar} which determines whether potentiation or depression occurs.

STDP alone can exhibit runaway dynamics which result in synaptic strengths saturating. In order to stabilize the performance of STDP, it is necessary to use the same synaptic scaling function used in the initialization step and intrinsic plasticity (Watt and Desai, 2010). Synaptic scaling normalizes the sum of pre-synaptic connections to α, as shown in (9).

$$W_{i,j} = \frac{W_{i,j}}{\sum_{j=1}^{N} W_{i,j}} * \alpha \tag{9}$$

Here, the synaptic connection from pre-synaptic neuron j to post-synaptic neuron i ($W_{i,j}$) is scaled so the total sum of the synaptic connections to neuron i remains constant. This helps stabilize the weights while maintaining the hebbian relation between synapses and removes the effect of noise on the network.

Global inhibition forces unsupervised learning through STDP to generate competition between neurons and causes neurons to learn different patterns. Global inhibition results in a winner-take-all network so that when a neuron fires to a specific pattern, it inhibits all other neurons from firing and learning that same pattern. To prevent a single neuron from constantly inhibiting other neurons, intrinsic plasticity (Watt and Desai, 2010) regulates how often a neuron fires by regulating the neurons firing threshold according to (10)

$$V_{th} = V_{th} + \Theta \tag{10}$$

where the neurons firing threshold V_{th} is increased by Θ and Θ is increased every time a neuron fires and decays back toward its resting value when a neuron does not fire according to a time constant τ shown in (11) (Zhang and Linden, 2003). The increased firing threshold decreases the probability of a neuron spiking multiple times in succession to allow other neurons to learn.

$$\tau \frac{d\Theta}{dt} = -\Theta \tag{11}$$

Unsupervised STDP with homeostatic mechanisms results in meaningful, low-dimensional representations of information present in the hidden layers utilizing only local plasticity mechanisms in contrast to training the entire network with expensive gradient descent based learning algorithms. This allows the deep-LSM to extract temporal information over

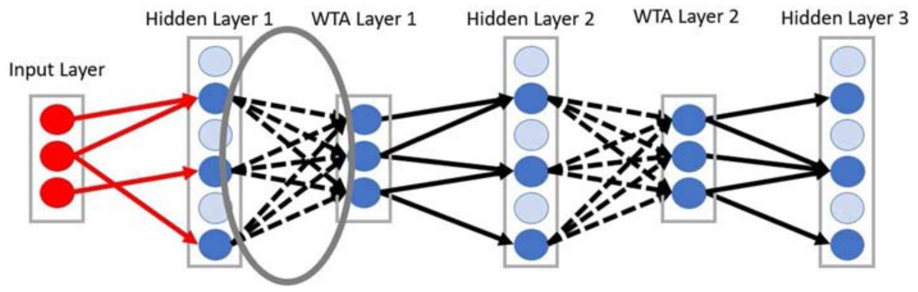


FIGURE 2 | Diagram of connectivity between layers in the deep-LSM. Dashed lines represent connections trained with STDP from a hidden layer to a WTA layer.

multiple time-scales with only local learning rules which is ideal for neuromorphic implementations (Neftci et al., 2017).

To summarize the information processing in the deep-LSM, the hidden layers capture dynamic information about the input signal over multiple times-scales. The WTA layers are trained to condense the high-dimensional hidden layer activity into a meaningful low-dimensional representation. This ensures that the inputs to each hidden layer provide useful information, while keeping the inputs to each hidden layer low-dimensional. This is important because the hidden layers rely on creating a high-dimensional representation of their input, by forming low-dimensional inputs it reduces the size of the deeper hidden layer which improves the scalability of the architecture.

3.3. Attention Mechanism

Another neural mechanism in the deep-LSM is the use of attention to selectively process information in the hidden layers as shown in **Figure 3**. As the size of the deep-LSM grows, attention allows the readout layer to perform classification with limited resources. Attention is applied by adding two separate single layer neural networks, which compute a weighted summation of all the hidden layers. This results in a single representation with the same dimensionality as one hidden layer being passed to the output layer. The attention networks receive the filtered state of the deep-LSM based on (4), $X_{deep-LSM} = [X_1, X_2, ..., X_{L-1}, X_L]$ where L is the number of hidden layers in the deep-LSM, to predict the appropriate attention coefficients.

First, the deep attention network predicts the importance of each layer in the deep-LSM. The attention network will predict a coefficient for each hidden layer in the deep-LSM based on the current state. The function of the deep attention network's operation is given by

$$A_l^{deep} = softmax_l(W^{A^{deep}} * X^{deep-LSM}) \tag{12}$$

where A_l refers to the attention coefficient for the l^{th} hidden layer in the network such that $A^{deep} = [A_1^{deep}, A_2^{deep}, ..., A_{L-1}^{deep}, A_L^{deep}]$ and L represents the total number of layers and $W^{A_l^{deep}}$ are the learned weights of the deep attention network. A softmax function is used to assign a probability to each layer which represents the importance of that layer. Then, based on the attention coefficients, a weighted sum of all the hidden layers is computed to generate a final representation of the deep-LSM (X^S) as shown in (13)

$$X^S = \sum_{l=1}^{L} A_l^{deep} * X_l \tag{13}$$

Second, the spatial attention network will predict the importance of each neuron in the final representation X_S. The second attention network receives the same input as the first attention network and will predict a coefficient $A_n^{Spatial}$ for every value in the final representation X^S, this can be applied to every neuron or a population of neurons. This will assign a weight to each neuron/population, allowing the output layer to focus on a select subset of signals. The operation for computing $A_n^{Spatial}$ is given by

$$A_n^{Spatial} = \sigma(W_n^{A^{Spatial}} * X_{deep-LSM}) \tag{14}$$

where each coefficient $A_n^{Spatial}$ is determined based on the learned weights for the n^{th} neuron in the spatial attention network, $W_n^{A^{spatial}}$, the state of the deep-LSM, and $A^{Spatial} = [A_1^{Spatial}, A_2^{Spatial}, ..., A_{N-1}^{Spatial}, A_N^{Spatial}]$ where N is the total number of neurons in a hidden layer. The coefficients in $A^{Spatial}$ will then be used to produce a weighted representation of X^S where

$$X^F = X^S \odot A^{Spatial} \tag{15}$$

where the final representation of deep-LSM's state X^F, is computed by an element-wise multiplication between the spatial attention coefficients and their corresponding location in X^S. X^F is then sent to the output layer which performs classification or prediction, given by (16)

$$y(n) = \sigma(W^{out} * X^F) \tag{16}$$

where $y(n)$ is the output of the readout layer based on the state H_F of the deep-LSM at time $t = n$.

4. EXPERIMENTS

The proposed deep-LSM was benchmarked for video activity recognition using the DogCentric dataset (Iwashita et al., 2014). The DogCentric dataset consists of 209 videos recorded for ten

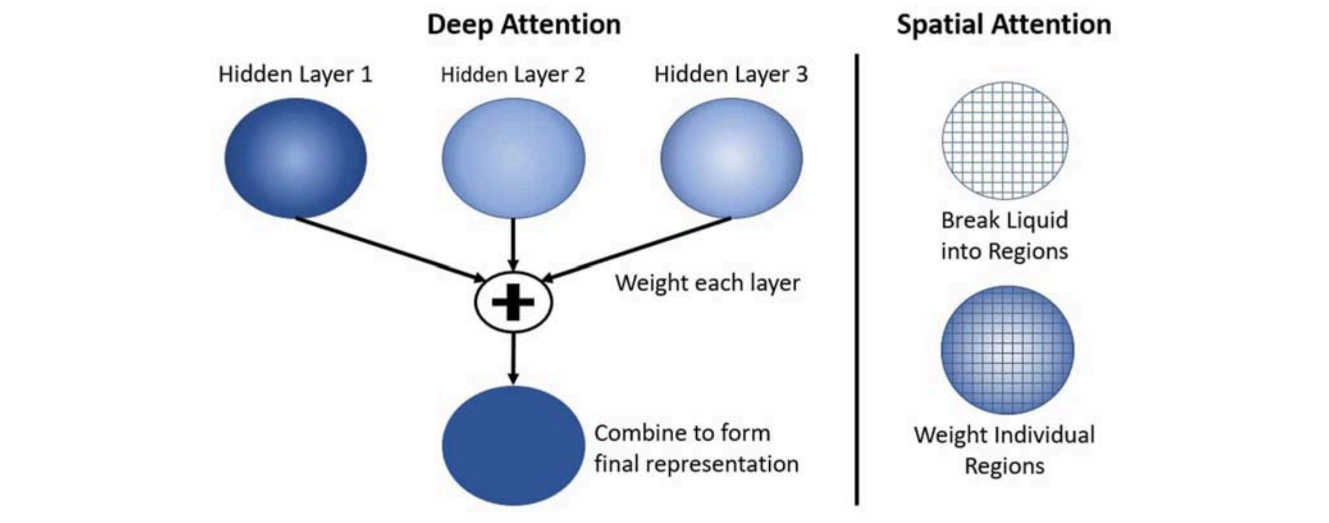


FIGURE 3 | Attention applied to the cumulative deep-LSM network referred to as the deep attention network **(Left)** and spatially to a single hidden layer through the spatial attention network **(Right)**.

FIGURE 4 | Sample video frame sequence from DogCentric dataset for the shake class [models tested with hand crafted features (HFC) and without HFCs are separated].

different activities being performed by four different dogs from a first-person view point. A sample of the image frames for the shake class is shown in **Figure 4**. The videos possess rapid and erratic movement, similar to a person running around with a camera, making it challenging to process what is occurring. There is also an imbalance in the datasamples with unequal number of videos per class. To make a fair comparison with prior networks, samples for every class were distributed equally between training and test data, similar to Graham et al. (2017).

The video frame features were extracted with a pre-trained ResNet-50 architecture which were then reduced to 100 dimensions using principal component analysis. The 100-dimensional features for each frame were used as an input to the deep-LSM and LSM models for classification at the end of each video sequence. The framelength in the DogCentric dataset varies from 30 frames to 650, with an average of 157 frames per video. Results were averaged for 150 runs of each model. The deep-LSM outperformed state-of-the-art models shown in **Table 1**, including a single layer LSM with an equal number of neurons and an attention modulated readout layer. The parameters used to obtain the results presented are given in **Table 2**.

To analyze the impact of different architectures in the deep-LSM, the network was studied for a different number of layers, for different sizes of the hidden layer, and for different sizes of the WTA layer. As shown in **Figure 5**, a single layer LSM is inferior to a deep-LSM with multiple layers and as the number of layers increases from three to five, the deep-LSM is better at processing the complex temporal information in the video.

The next analysis was how the size of the hidden layer affects performance shown in **Figure 6**. Increasing the size of the hidden layers or the WTA layers does not result in much difference in performance. For the size of the hidden layer, it is already sufficient with 1,000 neurons to create a high-dimensional representation of the input for extracting temporal information and further increases do not result in any change. If we decrease the hidden layer size, eventually a point is crossed where the high-dimensional representation does not capture enough information about the input and the performance will drop. This can be seen from the degradation in accuracy as the hidden layer size decreases to 250 neurons.

Lastly, **Figure 7** shows the performance as a function of the size of the WTA layer. For a 1,000 neuron hidden layer, increasing the WTA layer size from 50 to 100 neurons shows an increase in

TABLE 1 | Comparison of state-of-the-art accuracy results on the DogCentric dataset.

	Approach	Accuracy
HCF	GOFF + VIF + Log-C + Cuboids (Arabacı et al., 2018)	64.0%
	HOG+HOF+LBP+Cub.+Opt.Fl. (Iwashita et al., 2014)	60.5%
	ITF (Wang and Schmid, 2013; Piergiovanni et al., 2016)	67.7%
	ITF+CNN (Jain et al., 2014; Piergiovanni et al., 2016)	69.2%
	POT (Ryoo et al., 2015)	73.0%
	POT+ITF (Ryoo et al., 2015)	74.5%
	TDD (Wang et al., 2015; Piergiovanni et al., 2016)	76.6%
	TDD+Temp. Fil. (Piergiovanni et al., 2016)	79.6%
	TDD+Temp. Fil.+LSTM (Piergiovanni et al., 2016)	81.4%
No HCF	VGG+Max Pooling (Piergiovanni et al., 2016)	≈ 57.2%
	VGG+Mean Pooling (Piergiovanni et al., 2016)	59.9%
	VGG+Sum Pooling (Piergiovanni et al., 2016)	59.9%
	VGG+Temp. Fil.-Learned (Piergiovanni et al., 2016)	≈ 65.0%
	VGG+Temp. Fil.-Learned+LSTM (Piergiovanni et al., 2016)	≈ 65.0%
	CDN (VGG-16) (Graham et al., 2017)	75.8%
	CDN (ResNet-50) (Graham et al., 2017)	77.2%
	TCF (CaffeNet) (Kahani et al., 2017)	72.19%
	TCF (VGG-16) (Kahani et al., 2017)	77.79%
	TCS (VGG, TDD) (Kahani et al., 2017)	82.24%
	LFP (G+SD+GS) (Kwon et al., 2018)	82.5%
	Deep-LSM (ResNet-50)	**84.78%**
	LSM (ResNet-50)	76.5%

TABLE 2 | Parameters used in proposed deep-LSM and standard LSM implementation.

Parameter	Value
Simulation timestep	1 ms
V_{th}	16.5 mV
τ_m	28 ms
C_{mem}	1 pF
τ_{ref}	4 ms
D^H (deep-LSM/LSM)	1,000/3,000
E:I Ratio	4:1
Synaptic Strength (EE/EI/II/IE)	3/3/1/4
λ (2) (EE/EI/II/IE)	3/3/3/3
C (2) (EE/EI/II/IE)	0.6/1/0.2/1
α (9) (E/I)	40/36
α (3)	0.007
β (3)	0.739
D^W	50–100–150
X^{tar} (8)	25–50
τ_{trace} (4)	300 ms
α (9) (Synaptic scaling in WTA layer)	15
Θ (10)	1.5 mV
τ (11)	500 ms

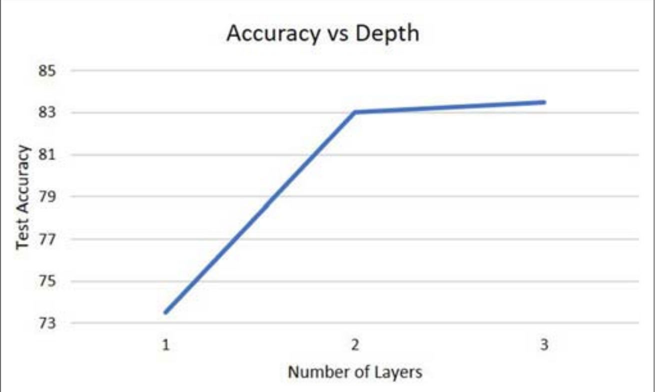


FIGURE 5 | Accuracy on the DogCentric dataset as a function of the number of layers in the deep-LSM (each hidden layer has 1,000 neurons, while each encoding layer has 50 neurons).

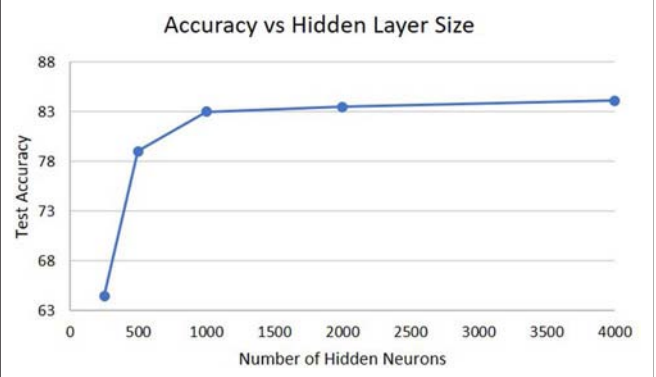


FIGURE 6 | Accuracy on the DogCentric dataset as a function of hidden layer size in a 3-layer deep-LSM (each WTA layer has 50 neurons).

performance because the WTA layer can capture more features describing the hidden layer. However, when the WTA layer size increases to 200 neurons the performance significantly drops. Similar results were observed for a 500 neuron hidden layer, which showed degradation in performance beyond 50 neurons. The reason for this is that there are now too many signals feeding into the next hidden layer which dominates the hidden layers dynamics, and because there is likely little information gained by the extra 100 neurons. We hypothesize that the optimal size of the WTA layer is dependent on the size of the hidden layer. With smaller hidden layers, there will be less features for the WTA layer to identify and learn so increasing the number of neurons does not have an impact on the information sent between layers. Another way to view this is as if one was doing principal component analysis on the hidden layers output, only the top few principal components would be needed to convey the important information between layers. In addition, the dimensionality of the WTA layer cannot be too close to the dimensionality of the hidden layer or it will negatively impact the information processing of deeper hidden layers. Another potential cause of this result is the hyper-parameters for the WTA layer are not optimal for allowing the network to efficiently learn at larger sizes (e.g., homeostatic mechanisms, training epochs).

4.1. Theoretical Efficiency for Neuromophic Implementations

To analyze the efficiency of the deep-LSM for on-device implementations, we study the deep-LSM in an application dependent framework for processing temporal information on embedded platforms. The first analysis is to compare the total number of synaptic connections as well as the types of training computations needed to assess the scalability and memory cost of the proposed model with respect to other recurrent neural networks. **Table 3** reports of the number of synaptic connections based on the type of learning for three temporal networks with an equal number of neurons; the deep-LSM, a traditional LSM, and an standard LSTM. A hypothetical LSTM model is used as a baseline purely for scalability analysis on the basis of an equivalent number of neurons and does not consider architectures such as stacked LSTMs. The analysis is performed for a deep-LSM which consists of 100 input neurons, 3 hidden layers with 500 neurons, two winner-take-all layers with 50 neurons, two attention networks (one with 3 neurons for each hidden layer and one with 500 neurons for each location in the hidden layer), and a readout layer with 10 neurons, one for each class. To determine the synaptic connections in the LSM and LSTM networks, we consider them to possess recurrent layers with 1500 neurons (which is equivalent to the total number of neurons in the three deep-LSM hidden layers). In addition, we consider a similar attention-based readout layer for the LSM which would implement spatial attention with 1500 neurons. As can be seen in **Table 3**, the deep-LSM with attention requires 35.69% of the number of synapses as the LSM with attention, but 613% the number of synapses as a standard LSM. However, a deep-LSM without attention only has 77.86% as many synapses as a standard LSM. In comparison to the LSTM model, a deep-LSM with the proposed attention mechanism has 8.87% of the number of synaptic connections with a similar number of neurons.

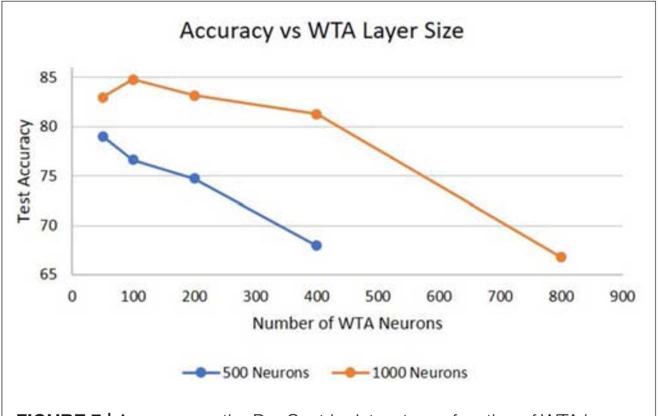


FIGURE 7 | Accuracy on the DogCentric dataset as a function of WTA layer size in a 3-layer deep-LSM (each hidden layer has 1,000 neurons).

From the table we can see that between the deep-LSM and LSM, with a similar readout layer (attention or single-layer), the deep-LSM shows a reduction in the number of synaptic weights. These calculations account for the sparsity values which had been used in our simulations, which was 95% sparsity in the input connections of both models, 89.24% sparsity in the deep-LSM hidden layers, and 95% sparsity in the LSM. Though the degree of sparsity varied in the hidden layer between the deep-LSM and LSM, they were generated from the same network hyper-parameters in (2). The difference arises from the deep-LSM having a smaller reservoir size which reduced the number of long-range connections which tended to not form a connection. In comparison to the LSTM, the deep-LSM with attention only has 7.93% as many trainable synaptic connections. In addition the deep-LSM attention weights are trained by a gradient descent algorithm which does not require sequential back-propagation-through-time. As for the connections trained through STDP, they only require an accumulation of a neurons activity (which is done per neuron rather than per synapse) and is only invoked when a neuron fires rather than every synapse being updated on each training operation. Therefore, the deep-LSM's training is computationally much lighter than the LSTM with respect to both the number and type of operations, and total number of trainable synapses.

The number of operations during inference and training in each model is reported in **Table 4**, which we computed for the deep-LSM and LSM based on our implementation, and for the LSTM based on derivation of the training and inference phase in Chen (2016) and are summarized in **Table 5** for inference and **Table 6** for training. These estimates calculate the number of multiplications needed in the specified models assuming that the number of additions would be similar and ignoring the cost of neuron functions and hyper-parameters. Based on the results, the deep-LSM with attention only has 8.45% of the number of computations as a vanilla LSTM and only 0.65% the number of computations without the attention module. In comparison between a deep-LSM and LSM, when an attention-based readout layer is used the deep-LSM has 64.84% fewer operations and significantly lower number of weight updates. Without attention the deep-LSM shows a 16.2% decrease but a slightly higher number of weight updates due to the unsupervised connections. Thus, separating the attention layer from the analysis, the deep-LSM shows a slight reduction in computational cost compared to the standard LSM.

Another important feature for algorithms on embedded platforms is robustness to device noise. To assess the robustness of the deep-LSM, we mimic device noise in a neuromemristive system by adding Gaussian noise on every read and write

TABLE 3 | Number of synaptic connections trained with different learning rules and their memory consumption for the deep-LSM, LSM, and LSTM.

	Backpropagation	Random	Unsupervised	Total	Memory (Gb)
Deep-LSM	15,000	90,738	2,500	108,238	0.0035
Deep-LSM + Attention	759,500	90,738	2,500	852,738	0.0273
LSM	15,000	124,016	0	139,016	0.0044
LSM + Attention	2,265,000	124,016	0	2,389,016	0.0764
LSTM	9,615,000	0	0	9,615,000	0.3077

TABLE 4 | Number of synaptic operations for a single frame of training data (for STDP synapses, only one post-synaptic neuron can win at any time frame).

Network	# Multiplications (FP)	# Multiplications (BP)	# Weight updates
Deep-LSM	110,700	15,000	17,500
Deep-LSM + Attention	857,200	773,506	760,500
LSM	135,000	15,000	15,000
LSM + Attention	2,386,500	2,251,500	2,251,500
LSTM	9,619,500	9,675,000	9,615,000

TABLE 5 | Computation of the number of multiplications needed during inference (FP).

Network	# Multiplications (Forward pass)
Deep-LSM	$S_{in} * N * H_d + 2(l-1) * S_{in} * (W * H_d) + l * S_R * (H_d * H_d) + l * H_d * O$
Deep-LSM + A	$S_{in} * N * H_d + 2(l-1) * S_{in} * (W * H_d) + l * S_R * (H_d * H_d) + l(H_d * A) + l * H_d + H_d + H_d * O$
LSM	$S_{in} * N * H + S_R * H * H + H * O$
LSM + A	$S_{in} * N * H + S_R * H * H + H * H + H * O$
LSTM	$4 * (N * H + H * H) + 3 * H + H * O$

N is the dimensionlaity of the input, H_d is the dimensionality of the deep-LSM hidden layers, W is the dimensionality of the WTA layers, A is the combined dimensionality of both attention networks, O is the dimensionality of the output, l is the number of layers, and H is the dimensionality of the hidden layer in the LSM and LSTM. For the LSM and $deep_L SM$, S_{in} is the input sparsity and S_R is the hidden layer sparsity. Note "+ A" refers to inclusion of the attention-based readout layer.

TABLE 6 | Computation of the number of multiplications needed during training (Backward Pass).

Network	# Multiplications (Backward pass)
Deep-LSM	$l * (O * H_d)$
Deep-LSM = A	$3 * (O * H_d) + A * H_d * l + 2 * H_d + 2 * l + l * 2 * H_d$
LSM	$H * O$
LSM + A	$H * O + H * H$
LSTM	$2 * (H * O) + 30 * H + 4 * (H * N) + 4 * (H * H)$

TABLE 7 | Performance on the DogCentrric dataset for a 3 layer deep-LSM when Gaussian noise is introduced.

Model (3 layers)	Accuracy	Standard deviation
Deep-LSM	82.9	6.78
Deep-LSM (with noise)	81.92	10.08

operation as in Soures et al. (2018). As shown in **Table 7**, the networks performance suffers very little degradation due to the presence of noise.

Finally, the energy consumption (estimated based on Han et al., 2016, for 45 nm technology node) of the proposed deep-LSM is compared with that of an LSM and LSTM. The energy is estimated by calculating the number of addition (0.9pJ) and multiplication (3.7pJ) operations (of 32-bit precision) for training and inference, and the number of synaptic weights stored in DRAM (360pJ).

Based on **Table 8**, it can be observed that the deep-LSM is more energy efficient than an LSTM during training, inference, and consumes less memory. When compared to the LSM, we see that the deep-LSM is more energy efficient when using an equivalent readout layer.

TABLE 8 | Energy portfolio of deep-LSM, LSM, and LSTM for inference, training, and memory.

	Inference Energy(μJ)	Training Energy (μJ)	Weights Energy (μJ)	Total Energy (μJ)
Deep-LSM	0.5092	0.069	38.9657	39.5439
Deep-LSM + Attention	3.9431	3.5581	306.9857	314.4869
LSM	0.621	0.069	50.0458	50.7358
LSM + Attention	10.9779	10.3569	860.0458	881.3806
LSTM	44.24	55.56	5866.6	5966.4

Estimates are for a 45 nm CMOS technology node (Han et al., 2016).

From this analysis, we conclude that the deep-LSM is a computationally lite model for processing temporal information with a fraction of the memory and compute operations compared to other popular recurrent neural network architectures. The deep-LSM has several features which result in its higher performance with respect to other algorithms. The first key feature of the deep-LSM is its modular reservoirs which create the deep architecture for the network. By using a modular approach, the deep-LSM reduces the size of the recurrent matrices needed by the network and also demonstrates a much better capability at extracting information over multiple time-scales as shown by the large increase in performance over traditional RC approaches. The second key feature of the deep-LSM is the use of spiking WTA layers in between hidden layers. This allows to extract meaningful features to propagate through the network and helps alleviate the dependence of traditional RC approaches on their initialization. The WTA layers learn their features through an unsupervised local learning rule which allows the network to learn and optimize its connections at a lower cost than gradient descent. Additionally, because STDP is a local learning rule the layers can be trained without waiting for information to be propagated backwards speeding up the training time and allowing the WTA layers to be updated in parallel. Finally, the last feature of the deep-LSM which contributes to its performance are the attention layers. Due to the large savings in total number of synaptic connections and reduced amount of training due to random connections, the deep-LSM can implement the attention layers while still maintaining an overall reduction in the number of synapses.

5. CONCLUSIONS

We proposed a new approach for performing spatio-temporal tasks on a budget. The proposed deep-LSM has promising results in video activity recognition achieving 84.78% on a representative dataset and surpasses state-of-the-art algorithms in accuracy. More importantly, the deep-LSM consumes significantly lower synaptic memory storage and computational resources. Edge devices naturally benefit from this computationally light algorithm and the following benefits ensue.

1. Edge intelligence framework: Suitable for real-time on-device learning and inference.
2. Local unsupervised plasticity mechanisms: Enable fine-grained tuning to trade-off compute complexity vs. accuracy.

3. Broaden applicability of RC approaches to complex temporal problems that require integration of information over multiple time-scales.
4. An overall reduction in energy consumption and memory requirements compared to current recurrent networks.

DATA AVAILABILITY

Publicly available datasets were analyzed in this study. This data can be found here: http://robotics.ait.kyushu-u.ac.jp/~yumi/db/first_dog.html.

AUTHOR CONTRIBUTIONS

NS as the first author performed the experiments and was responsible for writing and creating figures and tables. DK was responsible for writing and guidance in the design and experiments.

FUNDING

This material is based on research sponsored by AirForce Research Laboratory under agreement number FA8750-16-1-0108. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AirForce Research Laboratory or the U.S. Government.

978-1-5090-1370-8/16/$31.00 ©2016 Crown.

REFERENCES

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Hasan, M., Van Esesn, B. C., et al. (2018). The history began from alexnet: a comprehensive survey on deep learning approaches. *arXiv [Preprint]. arXiv: 1803.01164*. Available online at: https://arxiv.org/abs/1803.01164

Arabacı, M. A., Özkan, F., Surer, E., Jančovič, P., and Temizel, A. (2018). Multi-modal egocentric activity recognition using audio-visual features. *arXiv [Preprint]. arXiv: 1807.00612*. Available online at: https://arxiv.org/abs/1807.00612

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., and Baskurt, A. (2011). "Sequential deep learning for human action recognition," in *International Workshop on Human Behavior Understanding* (Berlin; Heidelberg: Springer), 29–39.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv [Preprint]. arXiv: 1803.09574*. Available online at: https://arxiv.org/abs/1803.09574

Carmichael, Z., Syed, H., Burtner, S., and Kudithipudi, D. (2018). "Mod-deepesn: modular deep echo state network," in *Conference on Cognitive Computational Neuroscience* (Philadelphia, PA).

Chen, G. (2016). A gentle tutorial of recurrent neural network with error backpropagation. *arXiv [Preprint]. arXiv: 1610.02583*. Available online at: https://arxiv.org/abs/1610.02583

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Fu, I., and Carter, C. (2016). *Benchmarking Training Time for CNN-based Detectors With Apache mxnet.* Available online at: https://aws.amazon.com/blogs/machine-learning/benchmarking-training-time-for-cnn-based-detectors-with-apache-mxnet/ (accessed November 16, 2018).

Gallicchio, C., and Micheli, A. (2016). "Deep reservoir computing: a critical analysis," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (Bruges).

Graham, D., Langroudi, S. H. F., Kanan, C., and Kudithipudi, D. (2017). "Convolutional drift networks for video classification," in *Rebooting Computing (ICRC), 2017 IEEE International Conference on IEEE* (Washington, DC), 1–8.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., et al. (2016). "Eie: efficient inference engine on compressed deep neural network," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on IEEE* (Seoul), 243–254.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Approach*. London: Psychology Press; John Wiley & Sons.

Hermans, M., and Schrauwen, B. (2013). "Training and analysing deep recurrent neural networks," in *Advances in Neural Information Processing Systems* (Stateline, NV), 190–198.

Iwashita, Y., Takamine, A., Kurazume, R., and Ryoo, M. S. (2014). "First-person animal activity recognition from egocentric videos," in *Pattern Recognition (ICPR), 2014 22nd International Conference on IEEE* (Stockholm), 4310–4315.

Jaeger, H. (2001). *The "echo state" Approach to Analysing and Training Recurrent Neural Networks-With an Erratum Note.* German National Research Center for Information Technology GMD Technical Report 148 (Bonn).

Jaeger, H. (2007). *Discovering Multiscale Dynamical Features With Hierarchical Echo State Networks*. Technical report, Jacobs University Bremen.

Jain, M., van Gemert, J., and Snoek, C. G. (2014). "University of amsterdam at thumos challenge 2014," in *ECCV THUMOS Challenge* (Zürich).

Kahani, R., Talebpour, A., and Mahmoudi-Aznaveh, A. (2017). A correlation based feature representation for first-person activity recognition. *arXiv [Preprint]. arXiv: 1711.05523*. doi: 10.1007/s11042-019-7429-3

Kwon, H., Kim, Y., Lee, J. S., and Cho, M. (2018). First person action recognition via two-stream convnet with long-term fusion pooling. *Patt. Recogn. Lett.* 112, 161–167. doi: 10.1016/j.patrec.2018.07.011

Lee, J., and Kim, J. (2016). Energy-efficient real-time human activity recognition on smart mobile devices. *Mobile Inform. Syst.* 2016, 1–12. doi: 10.1155/2016/2316757

Litwin-Kumar, A., Harris, K. D., Axel, R., Sompolinsky, H., and Abbott, L. (2017). Optimal degrees of synaptic connectivity. *Neuron* 93, 1153–1164. doi: 10.1016/j.neuron.2017.01.030

Ma, M., Fan, H., and Kitani, K. M. (2016). "Going deeper into first-person activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 1894–1903.

Ma, Q., Shen, L., and Cottrell, G. W. (2017). Deep-esn: a multiple projection-encoding hierarchical reservoir computing framework. *arXiv [Preprint]. arXiv: 1711.05255*. Available online at: https://arxiv.org/abs/1711.05255

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Markram, H., Wang, Y., and Tsodyks, M. (1998). Differential signaling via the same axon of neocortical pyramidal neurons. *Proc. Natl. Acad. Sci. U.S.A.* 95, 5323–5328. doi: 10.1073/pnas.95.9.5323

Neftci, E. O., Augustine, C., Paul, S., and Detorakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324

Piergiovanni, A., Fan, C., and Ryoo, M. S. (2016). Temporal attention filters for human activity recognition in videos. *arXiv [Preprint]. arXiv: 1605.08140.*

Possas, R., Caceres, S. P., and Ramos, F. (2018). "Egocentric activity recognition on a budget," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City), 5967–5976.

Renart, A., Song, P., and Wang, X.-J. (2003). Robust spatial working memory through homeostatic synaptic scaling in heterogeneous cortical networks. *Neuron* 38, 473–485. doi: 10.1016/S0896-6273(03)00255-1

Roy, S., and Basu, A. (2016). An online structural plasticity rule for generating better reservoirs. *Neural Comput.* 28, 2557–2584. doi: 10.1162/NECO_a_00886

Ryoo, M. S., Rothrock, B., and Matthies, L. (2015). "Pooled motion features for first-person videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA), 896–904.

Schrauwen, B., Verstraeten, D., and Van Campenhout, J. (2007). "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks* (Burges), 471–482.

Simonyan, K., and Zisserman, A. (2014). "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems* (Montreal, QC), 568–576.

Song, S., Chandrasekhar, V., Mandal, B., Li, L., Lim, J.-H., Sateesh Babu, G., et al. (2016a). "Multimodal multi-stream deep learning for egocentric activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (Las Vegas, NV), 24–31.

Song, S., Cheung, N.-M., Chandrasekhar, V., Mandal, B., and Liri, J. (2016b). "Egocentric activity recognition with multimodal fisher vector," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on IEEE* (Pudong), 2717–2721.

Soures, N., Hays, L., and Kudithipudi, D. (2017). "Robustness of a memristor based liquid state machine," in *Neural Networks (IJCNN), 2017 International Joint Conference on IEEE* (Anchorage, AK), 2414–2420.

Soures, N., Kudithipudi, D., Jacobs-Gedrim, R. B., Agarwal, S., and Marinella, M. (2018). Enabling on-device learning with deep spiking neural networks for speech recognition. *ECS Trans.* 85, 127–137. doi: 10.1149/08506.0127ecst

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). "Learning spatiotemporal9 features with 3d convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago), 4489–4497.

Triefenbach, F., Jalalvand, A., Demuynck, K., and Martens, J.-P. (2013). Acoustic modeling with hierarchical reservoirs. *IEEE Trans. Audio Speech Lang. Process.* 21, 2439–2450. doi: 10.1109/TASL.2013.2280209

Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J.-P. (2010). "Phoneme recognition with large hierarchical reservoirs," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 2307–2315.

Wang, H., and Schmid, C. (2013). "Action recognition with improved trajectories," in *Proceedings of the IEEE International Conference on Computer Vision* (Sydney, NSW), 3551–3558.

Wang, L., Qiao, Y., and Tang, X. (2015). "Action recognition with trajectory-pooled deep-convolutional descriptors," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA), 4305–4314.

Wang, Q., and Li, P. (2016). "D-lsm: deep liquid state machine with unsupervised recurrent reservoir tuning," in *Pattern Recognition (ICPR), 2016 23rd International Conference on IEEE* (Cancun), 2652–2657.

Watt, A. J., and Desai, N. S. (2010). Homeostatic plasticity and stdp: keeping a neuron's cool in a fluctuating world. *Front. Synaptic Neurosci.* 2:5. doi: 10.3389/fnsyn.2010.00005

Yan, Z., Subbaraju, V., Chakraborty, D., Misra, A., and Aberer, K. (2012). "Energy-efficient continuous activity recognition on mobile phones: an activity-adaptive approach," in *Wearable Computers (ISWC), 2012 16th International Symposium on IEEE* (Newcastle, UK), 17–24.

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). "Beyond short snippets: deep networks for video classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA), 4694–4702.

Zhan, K., Faux, S., and Ramos, F. (2014). "Multi-scale conditional random fields for first-person activity recognition," in *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on IEEE* (Budapest), 51–59.

Zhang, W., and Linden, D. J. (2003). The other side of the engram: experience-driven changes in neuronal intrinsic excitability. *Nat. Rev. Neurosci.* 4, 885–900. doi: 10.1038/nrn1248

Zheng, L., Wu, D., Ruan, X., Weng, S., Peng, A., Tang, B., et al. (2017). A novel energy-efficient approach for human activity recognition. *Sensors* 17:2064. doi: 10.3390/s17092064

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

TECHNOLOGY REPORT
published: 12 July 2019
doi: 10.3389/fnins.2019.00625

SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron

Milad Mozafari[1,2], Mohammad Ganjtabesh[1]*, Abbas Nowzari-Dalini[1] and Timothée Masquelier[2]

[1] Department of Computer Science, School of Mathematics, Statistics, and Computer Science, University of Tehran, Tehran, Iran, [2] CERCO UMR 5549, CNRS - Université Toulouse 3, Toulouse, France

OPEN ACCESS

Edited by:
Guoqi Li,
Tsinghua University, China

Reviewed by:
Deboleena Roy,
Purdue University, United States
Quansheng Ren,
Peking University, China

***Correspondence:**
Mohammad Ganjtabesh
mgtabesh@ut.ac.ir

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: 01 March 2019
Accepted: 31 May 2019
Published: 12 July 2019

Citation:
Mozafari M, Ganjtabesh M, Nowzari-Dalini A and Masquelier T (2019) SpykeTorch: Efficient Simulation of Convolutional Spiking Neural Networks With at Most One Spike per Neuron. Front. Neurosci. 13:625. doi: 10.3389/fnins.2019.00625

Application of deep convolutional spiking neural networks (SNNs) to artificial intelligence (AI) tasks has recently gained a lot of interest since SNNs are hardware-friendly and energy-efficient. Unlike the non-spiking counterparts, most of the existing SNN simulation frameworks are not practically efficient enough for large-scale AI tasks. In this paper, we introduce SpykeTorch, an open-source high-speed simulation framework based on PyTorch. This framework simulates convolutional SNNs with at most one spike per neuron and the rank-order encoding scheme. In terms of learning rules, both spike-timing-dependent plasticity (STDP) and reward-modulated STDP (R-STDP) are implemented, but other rules could be implemented easily. Apart from the aforementioned properties, SpykeTorch is highly generic and capable of reproducing the results of various studies. Computations in the proposed framework are tensor-based and totally done by PyTorch functions, which in turn brings the ability of just-in-time optimization for running on CPUs, GPUs, or Multi-GPU platforms.

Keywords: convolutional spiking neural networks, time-to-first-spike coding, one spike per neuron, STDP, reward-modulated STDP, tensor-based computing, GPU acceleration

1. INTRODUCTION

For many years, scientist were trying to bring human-like vision into machines and artificial intelligence (AI). In recent years, with advanced techniques based on deep convolutional neural networks (DCNNs) (Rawat and Wang, 2017; Gu et al., 2018), artificial vision has never been closer to human vision. Although DCNNs have shown outstanding results in many AI fields, they suffer from being data- and energy-hungry. Energy consumption is of vital importance when it comes to hardware implementation for solving real-world problems.

Our brain consumes much less energy than DCNNs, about 20 W (Mink et al., 1981) – roughly the power consumption of an average laptop, for its top-notch intelligence. This feature has convinced researchers to start working on computational models of human cortex for AI purposes. Spiking neural networks (SNNs) are the next generation of neural networks, in which neurons communicate through binary signals known as spikes. SNNs are energy-efficient for hardware implementation, because, spikes bring the opportunity of using event-based hardware as well as simple energy-efficient accumulators instead of complex energy-hungry multiply-accumulators that are usually employed in DCNN hardware (Furber, 2016; Davies et al., 2018).

Spatio-temporal capacity of SNNs makes them potentially stronger than DCNNs, however, harnessing their ultimate power is not straightforward. Various types of SNNs have been proposed for vision tasks which can be categorized based on their specifications such as:

- network structure: shallow (Masquelier and Thorpe, 2007; Yu et al., 2013; Kheradpisheh et al., 2016), and deep (Kheradpisheh et al., 2018; Mozafari et al., 2019),
- topology of connections: convolutional (Cao et al., 2015; Tavanaei and Maida, 2016), and fully connected (Diehl and Cook, 2015),
- information coding: rate (O'Connor et al., 2013; Hussain et al., 2014), and latency (Masquelier and Thorpe, 2007; Diehl and Cook, 2015; Mostafa, 2018),
- learning rule: unsupervised (Diehl and Cook, 2015; Ferré et al., 2018; Thiele et al., 2018), supervised (Diehl et al., 2015; Liu et al., 2017; Bellec et al., 2018; Mostafa, 2018; Shrestha and Orchard, 2018; Wu et al., 2018; Zenke and Ganguli, 2018), and reinforcement (Florian, 2007; Mozafari et al., 2018).

For recent advances in deep learning with SNNs, we refer the readers to reviews by Tavanaei et al. (2018), Pfeiffer and Pfeil (2018), and Neftci et al. (2019).

Deep convolutional SNNs (DCSNNs) with time-to-first-spike information coding and STDP-based learning rule constitute one of those many types of SNNs that carry interesting features. Their deep convolutional structure supports visual cortex and let them extract features hierarchically from simple to complex. Information coding using the earliest spike time, which is proposed based on the rapid visual processing in the brain (Thorpe et al., 1996), needs only a single spike, making them super fast and more energy efficient. These features together with hardware-friendliness of STDP, turn this type of SNNs into the best option for hardware implementation and online on-chip training (Yousefzadeh et al., 2017). Several recent studies have shown the excellence of this type of SNNs in visual object recognition (Kheradpisheh et al., 2018; Mostafa, 2018; Mozafari et al., 2018; Mozafari et al., 2019; Falez et al., 2019; Vaila et al., 2019).

With simulation frameworks such as Tensorflow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017), developing and running DCNNs is fast and efficient. Conversely, DCSNNs suffer from the lack of such frameworks. Existing state-of-the-art SNN simulators have been mostly developed for studying neuronal dynamics and brain functionalities and are not efficient and user-friendly enough for AI purposes. For instance, bio-realistic and detailed SNN simulations are provided by NEST (Gewaltig and Diesmann, 2007), BRIAN (Stimberg et al., 2014), NEURON (Carnevale and Hines, 2006), and ANNarchy (Vitay et al., 2015). These frameworks also enable users to define their own dynamics of neurons and connections. In contrast, frameworks such as Nengo (Bekolay et al., 2014) and NeuCube (Kasabov, 2014) offer high-level simulations focusing on the neural behavior of the network. Recently, BindsNet (Hazan et al., 2018) framework has been proposed as a fast and general SNN simulator based on PyTorch that is mainly developed for conducting AI experiments. A detailed comparison between BindsNet and the other available frameworks can be found in their paper.

In this paper, we propose SpykeTorch, a simulation framework based on PyTorch which is optimized specifically for convolutional SNNs with at most one spike per neuron. SpykeTorch offers utilities for building hierarchical feedforward SNNs with deep or shallow structures and learning rules such as STDP and R-STDP (Gerstner et al., 1996; Bi and Poo, 1998; Frémaux and Gerstner, 2016; Brzosko et al., 2017). SpykeTorch only supports time-to-first-spike information coding and provides a non-leaky integrate and fire neuron model with at most one spike per stimulus. Unlike BindsNet which is flexible and general, the proposed framework is highly restricted to and optimized for this type of SNNs. Although BindsNet is based on PyTorch, its network design language is different. In contrast, SpykeTorch is fully compatible and integrated with PyTorch and obeys the same design language. Therefore, a PyTorch user may only read the documentation to find out the new functionalities. Besides, this integrity makes it possible to utilize almost all of the PyTorch's functionalities either running on a CPU, or (multi-) GPU platform.

The rest of this paper is organized as follows: Section 2 describes how SpykeTorch includes the concept of time in its computations. Section 3 is dedicated to SpykeTorch package structure and its components. In section 4, a brief tutorial on building, training, and evaluating a DCSNN using SpykeTorch is given. Section 6 summarizes the current work and highlights possible future works.

2. TIME DIMENSION

Modules in SpykeTorch are compatible with those in PyTorch and the underlying data-type is simply the PyTorch's tensors. However, since the simulation of SNNs needs the concept of "time," SpykeTorch considers an extra dimension in tensors for representing time. The user may not need to think about this new dimensionality while using SpykeTorch, but, in order to combine it with other PyTorch's functionalities or extracting different kinds of information from SNNs, it is important to have a good grasp of how SpykeTorch deals with time.

SpykeTorch works with time-steps instead of exact time. Since the neurons emit at most one spike per stimulus, it is enough to keep track of the first spike times (in time-step scale) of the neurons. For a particular stimulus, SpykeTorch divides all of the spikes into a pre-defined number of spike bins, where each bin corresponds to a single time-step. More precisely, assume a stimulus is represented by F feature maps, each constitutes a grid of $H \times W$ neurons. Let T_{max} be the maximum possible number of time-steps (or bins) and $T_{f,r,c}$ denote the spike time (or the bin index) of the neuron placed at position (r, c) of the feature map f, where $0 \leq f < F$, $0 \leq r < H$, $0 \leq c < W$, and $T_{f,r,c} \in \{0, 1, ..., T_{max} - 1\} \cup \{\infty\}$. The ∞ symbol stands for no spike. SpykeTorch considers this stimulus as a four-dimensional binary spike-wave tensor S of size $T_{max} \times F \times H \times W$ where:

$$S[t, f, r, c] = \begin{cases} 0 & t < T_{f,r,c}, \\ 1 & otherwise. \end{cases} \qquad (1)$$

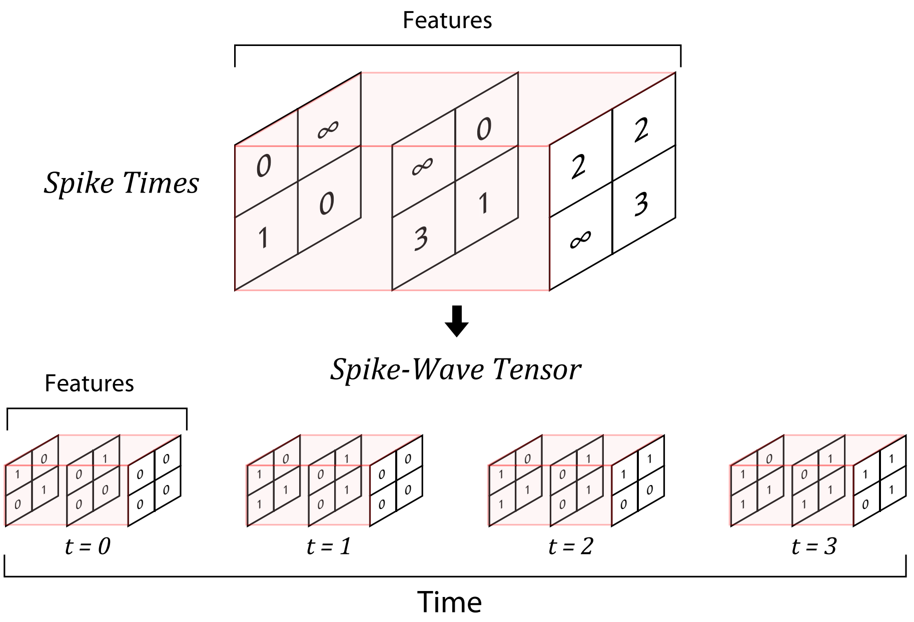


FIGURE 1 | An example of generating spike-wave tensor from spike times. There are three feature maps, each constitutes a 2 × 2 grid of neurons that are going to generate a spike-wave representing a stimulus. If the maximum number of time-steps is 4, then the resulting spike-wave is a four-dimensional tensor of size 4 × 3 × 2 × 2. If a neuron emits spike at time step $t = i$, the corresponding position in the spike-wave tensor will be set to 1 from time-step $t = i$ to the final time step ($t = 3$).

Note that this way of keeping the spikes (accumulative structure) does not mean that neurons keep firing after their first spikes. Repeating spikes in future time steps increases the memory usage, but makes it possible to process all of the time-steps simultaneously and produce the corresponding outputs, which consequently results in a huge speed-up. **Figure 1** illustrates an example of converting spike times into a SpykeTorch-compatible spike-wave tensor. **Figure 2** shows how accumulative spikes helps simultaneous computations.

3. PACKAGE STRUCTURE

Basically, SpykeTorch consists of four python modules; (1) `snn` which contains multiple classes for creating SNNs, (2) `functional` that implements useful SNNs' functions, (3) `utils` which gathers helpful utilities, and (4) `visualization` which helps to generate graphical data out of SNNs. The following subsections explain these modules.

3.1. snn Module

The `snn` module contains necessary classes to build SNNs. These classes are inherited from the PyTorch's `nn.Module`, enabling them to function inside the PyTorch framework as network modules. Since we do not support error backpropagation, the PyTorch's auto-grad feature is turned off for all of the parameters in `snn` module.

`snn.Convolutional` objects implements spiking convolutional layers with two-dimensional convolution kernel. A `snn.Convolutional` object is built by providing the number of input and output features (or channels), and the size of the convolution kernel. Given the size of the kernel, the corresponding tensor of synaptic weights is randomly initialized using a normal distribution, where the mean and standard deviation can be set for each object, separately.

A `snn.Convolutional` object with kernel size $K_h \times K_w$ performs a valid convolution (with no padding) over an input spike-wave tensor of size $T_{max} \times F_{in} \times H_{in} \times W_{in}$ with stride equals to 1 and produces an output potentials tensor of size $T_{max} \times F_{out} \times H_{out} \times W_{out}$, where:

$$\begin{aligned} H_{out} &= H_{in} - K_h + 1, \\ W_{out} &= W_{in} - K_w + 1, \end{aligned} \tag{2}$$

and F_{in} and F_{out} are the number of input and output features, respectively. Potentials tensors (P) are similar to the binary spike-wave tensors, however $P[t, f, r, c]$ denotes the floating-point potential of a neuron placed at position (r, c) of feature map f, at time-step t. Note that current version of SpykeTorch does not support stride more than 1, however, we are going to implement it in the next major version.

The underlying computation of `snn.Convolutional` is the PyTorch's two-dimensional convolution, where the mini-batch dimension is sacrificed for the time. According to the accumulative structure of spike-wave tensor, the result of applying PyTorch's `conv2D` over this tensor is the accumulative potentials over time-steps.

It is important to mention that simultaneous computation over time dimension improves the efficiency of the framework, but it has dispelled batch processing in SpykeTorch. We agree that batch processing brings a huge speed-up, however, providing it to the current version of SpykeTorch is not straightforward. Here are some of the important challenges: (1) Due to accumulative format of spike-wave tensors, keeping batch of images increases memory usage even more. (2) Plasticity in batch mode needs new strategies. (3) To get the most out of batch processing, all of the main computations such as plasticity, competitions, and inhibitions should be done on the whole batch at the same time, especially when the model is running on GPUs.

Pooling is an important operation in deep convolutional networks. `snn.Pooling` implements two-dimensional max-pooling operation. Building `snn.Pooling` objects requires providing the pooling window size. The stride is equal to the window size by default, but it is adjustable. Zero padding is also another option which is off by default.

`snn.Pooling` objects are applicable to both spike-wave and potentials tensors. According to the structure of these tensors, if the input is a spike-wave tensor, then the output will contain the earliest spike within each pooling window, while if the input is a potentials tensor, the maximum potential within each pooling window will be extracted. Assume that the input tensor has the shape $T_{max} \times F_{in} \times H_{in} \times W_{in}$, the pooling window has the size $P_h \times P_w$ with stride $R_h \times R_w$, and the padding is (D_h, D_w), then the output tensor will have the size $T_{max} \times F_{out} \times H_{out} \times W_{out}$, where:

$$\begin{aligned} H_{out} &= \lfloor \frac{H_{in} + 2 \times D_h}{R_h} \rfloor, \\ W_{out} &= \lfloor \frac{W_{in} + 2 \times D_w}{R_w} \rfloor. \end{aligned} \tag{3}$$

To apply STDP on a convolutional layer, a `snn.STDP` object should be built by providing the value of required parameters such as learning rates. Since this simulator works with time-to-first-spike coding, the provided implementation of the STDP function is as follows:

$$\Delta \mathcal{W}_{i,j} = \begin{cases} A^{+} \times (\mathcal{W}_{i,j} - LB) \times (UB - \mathcal{W}_{i,j}) & \text{if} \quad T_j \leq T_i, \\ A^{-} \times (\mathcal{W}_{i,j} - LB) \times (UB - \mathcal{W}_{i,j}) & \text{if} \quad T_j > T_i, \end{cases} \tag{4}$$

where, $\Delta \mathcal{W}_{i,j}$ is the amount of weight change of the synapse connecting the post-synaptic neuron i to the pre-synaptic neuron j, A^{+} and A^{-} are learning rates, and $(\mathcal{W}_{i,j} - LB) \times (UB - \mathcal{W}_{i,j})$ is a stabilizer term which slows down the weight change when the synaptic weight $(\mathcal{W}_{i,j})$ is close to the lower (LB) or upper (UB) bounds.

To apply STDP during the training process, providing the input and output spike-wave, as well as output potentials tensors are necessary. `snn.STDP` objects make use of the potentials tensor to find winners. Winners are selected first based on the earliest spike times, and then based on the maximum potentials. The number of winners is set to 1 by default. `snn.STDP` objects also provide lateral inhibition, by which they completely inhibit the winners' surrounding neurons in all of the feature maps within a specific distance. This increases the chance of learning diverse features. Note that R-STDP can be applied using two `snn.STDP` objects; one for STDP part and the other for anti-STDP part.

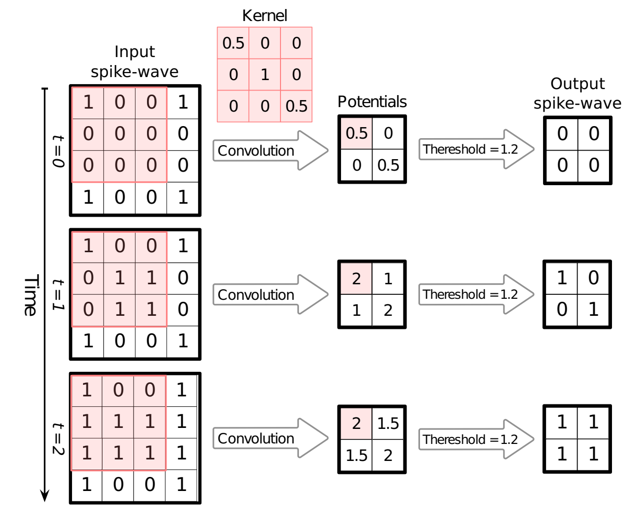


FIGURE 2 | An example of simultaneous processing of spikes over time-steps. Here the input spike-wave tensor has one 5 × 5 channel (feature map) and the spikes are divided into three time-steps. When SpykeTorch applies the convolution kernel of size 3 × 3 (valid mode) simultaneously on all of the time-steps, the resulting tensor will contain potentials in all of the time-steps. Since spikes are stored in accumulative format, then the potentials are accumulative as well. Applying a threshold function over the whole potential tensor generates the corresponding output spike-wave tensor, again in accumulative format.

3.2. `functional` Module

This module contains several useful and popular functions applicable on SNNs. Here we briefly review the most important ones. For the sake of simplicity, we replace the `functional.` with `sf.` for writing the function names.

As mentioned before, `snn.Convolutional` objects give potential tensors as their outputs. `sf.fire` takes a potentials tensor as input and converts it into a spike-wave tensor based on a given threshold. `sf.threshold` function is also available separately that takes a potentials tensor and outputs another potentials tensor in which all of the potentials lower than the given threshold are set to zero. The output of `sf.threshold` is called thresholded potentials.

Lateral inhibition is another vital operation for SNNs specially during the training process. It helps to learn more diverse features and achieve sparse representations in the network. SpykeTorch's `functional` module provides several functions for different kinds of lateral inhibitions.

`sf.feature_inhibition` is useful for complete inhibition of the selected feature maps. This function comes in handy to apply dropout to a layer. `sf.pointwise_inhibition` employs competition among features. In other words, at each location, only the neuron corresponding to the most salient feature will be allowed to emit a spike (the earliest spike with the highest potential). Lateral inhibition is also helpful to be applied on input intensities before conversion to spike-waves. This will decrease the redundancy in each region of the input. To apply this kind of inhibition, `sf.intensity_lateral_inhibition` is provided. It takes intensities and a lateral inhibition kernel by which it decreases the surrounding intensities (thus increases the latency of the corresponding spike) of each salient point. Local normalization is also provided by `sf.local_normalization` which uses regional mean for normalizing intensity values.

Winners-take-all (WTA) is a popular competition mechanism in SNNs. WTA is usually used for plasticity, however, it can be involved in other functionalities such as decision-making. `sf.get_k_winners` takes the desired number of winners and the thresholded potentials and returns the list of winners. Winners are selected first based on the earliest spike times, and then based on the maximum potentials. Each winner's location is represented by a triplet of the form (*feature*, *row*, *column*).

3.3. `utils` Module

`utils` module provides several utilities to ease the implementation of ideas with SpykeTorch. For example, `utils.generate_inhibition_kernel` generates an inhibition kernel based on a series of inhibition factors in a form that can be properly used by `sf.intensity_lateral_inhibition`.

There exist several transformation utilities that are suitable for filtering inputs and converting them to spike-waves. Current utilities are mostly designed for vision purposes. `utils.LateralIntencityInhibition` objects do the `sf.intensity_lateral_inhibition` as a transform object. `utils.FilterKernel` is a base class to define filter kernel generators. SpykeTorch has already provided `utils.DoGKernel` and `utils.GaborKernel` in order to generate DoG and Gabor filter kernels, respectively. Objects of `utils.FilterKernel` can be packed into a multi-channel filter kernel by `utils.Filter` objects and applied to the inputs.

The most important utility provided by `utils` is `utils.Intensity2Latency`. Objects of `utils.Intensity2Latency` are used as transforms in PyTorch's datasets to transform intensities into latencies, i.e., spike-wave tensors. Usually, `utils.Intensity2Latency` is the final transform applied to inputs.

Since the application of a series of transformations and the conversion to spike-waves can be time-consuming, SpykeTorch provides a wrapper class, called `utils.CacheDataset`, which is inherited from PyTorch's `dataset` class. Objects of `utils.CacheDataset` take a dataset as their input and cache the data after applying all of the transformations. They can cache either on primary memory or secondary storage.

Additionally, `utils` contains two functions `utils.tensor_to_text` and `utils.text_to_tensor`, which handle conversion of tensors to text files and the reverse, respectively. This conversion is helpful to import data from a source or export a tensor for a target software. The format of the text file is as follows: the first line contains comma-separated integers denoting the shape of the tensor. The second line contains comma-separated values indicating the whole tensor's data in row-major order.

3.4. `visualization` Module

The ability to visualize deep networks is of great importance since it gives a better understanding of how the network's components are working. However, visualization is not a straightforward procedure and depends highly on the target problem and the input data.

Due to the fact that SpykeTorch is developed mainly for vision tasks, its `visualization` module contains useful functions to reconstruct the learned visual features. The user should note that these functions are not perfect and cannot be used in every situation. In general, we recommend the user to define his/her own visualization functions to get the most out of the data.

4. TUTORIAL

In this section, we show how to design, build, train, and test a SNN with SpykeTorch in a tutorial format. The network in this tutorial is adopted from the deep convolutional SNN proposed by Mozafari et al. (2019) which recognizes handwritten digits (tested on MNIST dataset). This network has a deep structure and uses both STDP and Reward-Modulated STDP (R-STDP), which makes it a suitable choice for a complete tutorial. In order to make the tutorial as simple as possible, we present code snippets with reduced contents. For the complete source code, please check SpykeTorch's GitHub[1] web page.

4.1. Step 1. Network Design

4.1.1. Structure

The best way to design a SNN is to define a class inherited from `torch.nn.Module`. The network proposed by Mozafari et al. (2019), has an input layer which applies DoG filters to the input image and converts it to spike-wave. After that, there are three convolutional (S) and pooling (C) layers that are arranged in the form of $S1 \rightarrow C1 \rightarrow S2 \rightarrow C2 \rightarrow S3 \rightarrow C3$ (see **Figure 3**). Therefore, we need to consider three objects for convolutional layers in this model. For the pooling layers, we will use the functional version instead of the objects.

As shown in **Listing 1**, three `snn.Convolutional` objects are created with desired parameters. Two `snn.STDP` objects are built for training $S1$ and $S2$ layers. Since $S3$ is trained by R-STDP, two `snn.STDP` are needed to cover both STDP and anti-STDP parts. To have the effect of anti-STDP, it is enough to negate the signs of the learning rates. Note that the `snn.STDP` objects for `conv3` have two extra parameters where the first one turns off the stabilizer and the second one keeps the weights in range $[0.2, 0.8]$.

Although `snn` objects are compatible with `nn.Sequential` (`nn.Sequential` automates the forward pass given the network modules), we cannot use it at the moment. The reason is that different learning rules may need different kinds of data from each layer, thus accessing each layer during the forward pass is a must.

[1]https://github.com/miladmozafari/SpykeTorch

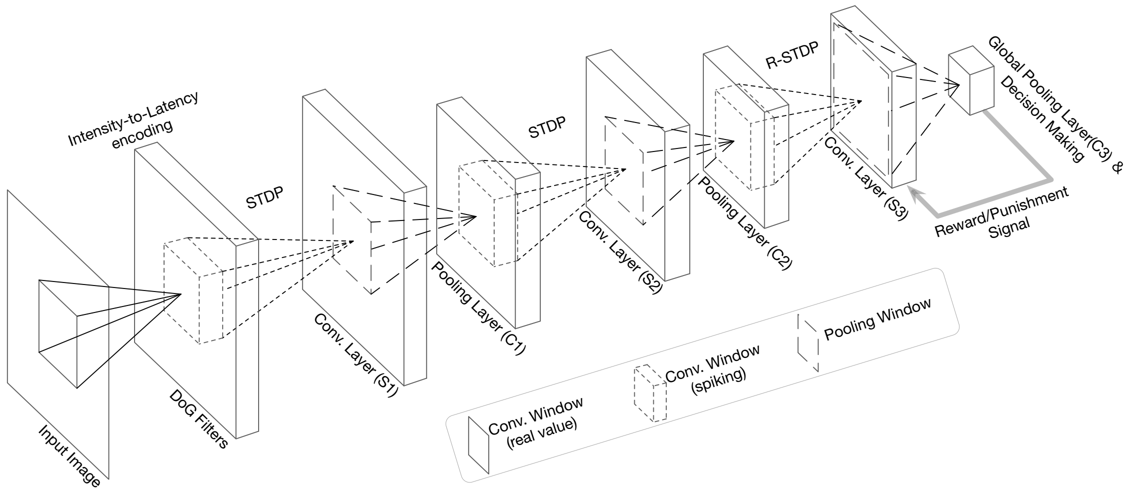


FIGURE 3 | Overall structure of the network in the tutorial (modified figure from the work by Mozafari et al., 2019).

```python
import torch.nn as nn
import SpykeTorch.snn as snn
import SpykeTorch.functional as sf
class DCSNN(nn.Module):
   def __init__(self):
      super(DCSNN, self).__init__()

      #(in_channels, out_channels, kernel_size, weight_mean=0.8, weight_std=0.02)
      self.conv1 = snn.Convolution(6, 30, 5, 0.8, 0.05)
      self.conv2 = snn.Convolution(30, 250, 3, 0.8, 0.05)
      self.conv3 = snn.Convolution(250, 200, 5, 0.8, 0.05)

      #(conv_layer, learning_rate, use_stabilizer=True, lower_bound=0, upper_bound=1)
      self.stdp1 = snn.STDP(self.conv1, (0.004, -0.003))
      self.stdp2 = snn.STDP(self.conv2, (0.004, -0.003))
      self.stdp3 = snn.STDP(self.conv3, (0.004, -0.003), False, 0.2, 0.8)
      self.anti_stdp3 = snn.STDP(self.conv3, (-0.004, 0.0005), False, 0.2, 0.8)
```

Listing 1 | Defining the network class.

4.1.2. Forward Pass

Next, we implement the forward pass of the network. To this end, we override the `forward` function in `nn.Module`. If the training is off, then implementing the forward pass will be straightforward. **Listing 2** shows the application of convolutional and pooling layers on an input sample. Note that each input is a spike-wave tensor. We will demonstrate how to convert images into spike-waves later.

As shown in **Listing 2**, the input of each convolutional layer is the padded version of the output of its previous layer, thus, there would be no information loss at the boundaries. Pooling operations are also applied by the corresponding function `sf.pooling`, which is an alternative to `snn.Pooling`. According to Mozafari et al. (2019), their proposed network makes decision based on the maximum potential among the neurons in the last pooling layer. To this end, we use an infinite threshold for the last convolutional layer by omitting its value from `sf.fire_` function. `sf.fire_` is the in-place version of `sf.fire` which modifies the input potentials tensor P_{in} as follows:

$$P_{in}[t,f,r,c] = \begin{cases} 0 & \text{if } \ t < T_{max} - 1, \\ P_{in}[t,f,r,c] & \text{otherwise.} \end{cases} \tag{5}$$

Consequently, the resulting spike-wave will be a tensor in which, all the values are zero except for those non-zero potential values in the last time-step.

Now that we have the potentials of all the neurons in $S3$, we find the only one winner among them. This is the same as doing a global max-pooling and choosing the maximum potential among

```python
def forward(self, input):
   input = sf.pad(input, (2,2,2,2))
   if not self.training:
      pot = self.conv1(input)
      spk = sf.fire(pot, 15)
      pot = self.conv2(sf.pad(sf.pooling(spk, 2, 2), (1,1,1,1)))
      spk = sf.fire(pot, 10)
      pot = self.conv3(sf.pad(sf.pooling(spk, 3, 3), (2,2,2,2)))
      # omitting the threshold parameters means infinite threshold
      spk = sf.fire_(pot)
      winners = sf.get_k_winners(pot, 1)
      output = -1
      # each winner is a tuple of form (feature, row, column)
      if len(winners) != 0:
         output = self.decision_map[winners[0][0]]
      return output
```

Listing 2 | Defining the forward pass (during testing process).

```python
def save_data(self, input_spk, pot, spk, winners):
   self.ctx["input_spikes"] = input_spk
   self.ctx["potentials"] = pot
   self.ctx["output_spikes"] = spk
   self.ctx["winners"] = winners
```

Listing 3 | Saving required data for plasticity.

them. `decision_map` is a Python list which maps each feature to a class label. Since each winner contains the feature number as its first component, we can easily indicate the decision of the network by putting that into the `decision_map`.

We cannot take advantage of this forward pass during the training process as the STDP and R-STDP need local synaptic data to operate. Therefore, we need to save the required data during the forward pass. We define a Python dictionary (named `ctx`) in our network class and a function which saves the data into that (see **Listing 3**). Since the training process is layer-wise, we update the `forward` function to take another parameter which specifies the layer that is under training. The updated `forward` function is shown in **Listing 4**.
There are several differences with respect to the testing forward pass. First, `sf.fire` is used with an extra parameter value. If the value of this parameter is `True`, the tensor of thresholded potentials will also be returned. Second, `sf.get_k_winners` is called with a new parameter value which controls the radius of lateral inhibition. Third, the forward pass is interrupted by the value of `max_layer`.

4.1.3. Plasticity

Now that we saved the required data for learning, we can define a series of helper functions to apply STDP or anti-STDP. **Listing 5** defines three member functions for this purpose. For each call of `STDP` objects, we need to provide tensors of input spike-wave, output thresholded potentials, output spike-wave, and the list of winners.

4.2. Step 2. Input Layer and Transformations

SNNs work with spikes, thus, we need to transform images into spike-waves before feeding them into the network. PyTorch's datasets accept a function as a transformation which is called automatically on each input sample. We make use of this feature together with the provided transform functions and objects by PyTorch and SpykeTorch. According to the network proposed by Mozafari et al. (2019), each image is convolved by six DoG filters, locally normalized, and transformed into spike-wave. As appeared in **Listing 6**, a new class is defined to handle the required transformations.

Each `InputTransform` object converts the input image into a tensor (line 9), adds an extra dimension for time (line 10), applies provided filters (line 11), applies local normalization (line 12), and generates spike-wave tensor (line 13). To create `utils.Filter` object, six DoG kernels with desired parameters are given to `utils.Filter`'s constructor (lines 15–17) as well as an appropriate padding and threshold value (line 18).

```python
def forward(self, input, max_layer):
    input = sf.pad(input, (2,2,2,2))
    if self.training: #forward pass for train
        pot = self.conv1(input)
        spk, pot = sf.fire(pot, 15, True)
        if max_layer == 1:
            winners = sf.get_k_winners(pot, 5, 3)
            self.save_data(input, pot, spk, winners)
            return spk, pot
        spk_in = sf.pad(sf.pooling(spk, 2, 2), (1,1,1,1))
        pot = self.conv2(spk_in)
        spk, pot = sf.fire(pot, 10, True)
        if max_layer == 2:
            winners = sf.get_k_winners(pot, 8, 2)
            self.save_data(spk_in, pot, spk, winners)
            return spk, pot
        spk_in = sf.pad(sf.pooling(spk, 3, 3), (2,2,2,2))
        pot = self.conv3(spk_in)
        spk = sf.fire_(pot)
        winners = sf.get_k_winners(pot, 1)
        self.save_data(spk_in, pot, spk, winners)
        output = -1
        if len(winners) != 0:
            output = self.decision_map[winners[0][0]]
        return output
    else:
        # forward pass for testing process
```

Listing 4 | Defining the forward pass (during training process).

```python
def stdp(self, layer_idx):
    if layer_idx == 1:
        self.stdp1(self.ctx["input_spikes"], self.ctx["potentials"],
        ↪  self.ctx["output_spikes"], self.ctx["winners"])
    if layer_idx == 2:
        self.stdp2(self.ctx["input_spikes"], self.ctx["potentials"],
        ↪  self.ctx["output_spikes"], self.ctx["winners"])

def reward(self):
    self.stdp3(self.ctx["input_spikes"], self.ctx["potentials"], self.ctx["output_spikes"],
    ↪  self.ctx["winners"])

def punish(self):
    self.anti_stdp3(self.ctx["input_spikes"], self.ctx["potentials"],
    ↪  self.ctx["output_spikes"], self.ctx["winners"])
```

Listing 5 | Defining helper functions for plasticity.

4.3. Step 3. Data Preparation

Due to the PyTorch and SpykeTorch compatibility, all of the PyTorch's dataset utilities work here. As illustrated in **Listing 7**, we use `torchvision.datasets.MNIST` to load MNIST dataset with our previously defined `transform`. Moreover, we use SpykeTorch's dataset wrapper, `utils.CacheDataset` to enable caching the transformed data after its first presentation. When the dataset gets ready, we use PyTorch's `DataLoader` to manage data loading.

4.4. Step 4. Training and Testing

4.4.1. Unsupervised Learning (STDP)

To do unsupervised learning on *S1* and *S2* layers, we use a helper function as defined in **Listing 8**. This function trains

```python
import SpykeTorch.utils as utils
import torchvision.transforms as transforms
class InputTransform:
   def __init__(self, filter):
      self.to_tensor = transforms.ToTensor()
      self.filter = filter
      self.temporal_transform = utils.Intensity2Latency(15, to_spike=True)
   def __call__(self, image):
      image = self.to_tensor(image) * 255
      image.unsqueeze_(0)
      image = self.filter(image)
      image = sf.local_normalization(image, 8)
      return self.temporal_transform(image)

kernels = [ utils.DoGKernel(3,3/9,6/9), utils.DoGKernel(3,6/9,3/9),
         utils.DoGKernel(7,7/9,14/9), utils.DoGKernel(7,14/9,7/9),
         utils.DoGKernel(13,13/9,26/9), utils.DoGKernel(13,26/9,13/9)]
filter = utils.Filter(kernels, padding = 6, thresholds = 50)
transform = InputTransform(filter)
```

Listing 6 | Transforming each input image into spike-wave.

```python
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
MNIST_train = utils.CacheDataset(MNIST(root=data_root, train=True, download=True,
 ↪  transform=transform))
MNIST_test = utils.CacheDataset(MNIST(root=data_root, train=False, download=True,
 ↪  transform=transform))
MNIST_loader = DataLoader(MNIST_train, batch_size=1000)
MNIST_test_loader = DataLoader(MNIST_test, batch_size=len(MNIST_test))
```

Listing 7 | Preparing MNIST dataset and the data loader.

```python
def train_unsupervised(network, data, layer_idx):
   network.train()
   for i in range(len(data)):
      data_in = data[i].cuda() if use_cuda else data[i]
      network(data_in, layer_idx)
      network.stdp(layer_idx)
```

Listing 8 | Helper function for unsupervised learning.

layer `layer_idx` of `network` on `data` by calling the corresponding STDP object. There are two important things in this function: (1) putting the network in train mode by calling. `train` function, and (2) loading the sample on GPU if the global `use_cuda` flag is `True`.

4.4.2. Reinforcement Learning (R-STDP)

To apply R-STDP, it is enough to call previously defined `reward` or `punish` member functions under appropriate conditions. As shown in **Listing 9**, we check the network's decision with the label and call `reward` (or `punish`) if it matches (or mismatches) the target. We also compute the performance by counting correct, wrong, and silent (no decision is made because of receiving no spikes) samples.

4.4.3. Execution

Now that we have the helper functions, we can make an instance of the network and start training and testing it. **Listing 10** illustrates the implementation of this part. Note that the `test` helper function is the same as the `train_rl` function, but it

```python
import numpy as np
def train_rl(network, data, target):
   network.train()
   perf = np.array([0,0,0]) # correct, wrong, silent
   for i in range(len(data)):
      data_in = data[i].cuda() if use_cuda else data[i]
      target_in = target[i].cuda() if use_cuda else target[i]
      d = network(data_in, 3)
      if d != -1:
         if d == target_in:
            perf[0]+=1
            network.reward()
         else:
            perf[1]+=1
            network.punish()
      else:
         perf[2]+=1
   return perf/len(data)
```

Listing 9 | Helper function for reinforcement learning.

```python
net = DCSNN()
if use_cuda:
   net.cuda()
# First Layer
for epoch in range(epochs_1):
   for data,targets in MNIST_loader:
      train_unsupervised(net, data, 1)
# Second Layer
for epoch in range(epochs_2):
   for data,targets in MNIST_loader:
      train_unsupervised(net, data, 2)
# Third Layer
for epoch in range(epochs_3):
   for data,targets in MNIST_loader: # Training
      print(train_rl(net, data, targets))
   for data,targets in MNIST_test_loader: # Testing
      print(test(net, data, targets))
```

Listing 10 | Training and testing the network.

calls `network.eval` instead of `network.train` and it does not call plasticity member functions. Also, invoking `net.cuda`, transfers all the network's parameters to the GPU.

4.5. Source Code

Through this tutorial, we omitted many parts of the actual implementation such as adaptive learning rates, multiplication of learning rates, and saving/loading the best state of the network, for the sake of simplicity and clearance. The complete reimplementation is available on SpykeTorch's GitHub web page. We have also provided scripts for other works (Kheradpisheh et al., 2018; Mozafari et al., 2018) that achieve almost the same results as the main implementations. However, due to technical and computational differences between SpykeTorch and the original versions, tiny differences in performance are expected. A comparison between SpykeTorch and one of the previous implementations is provided in the next section.

5. COMPARISON

We performed a comparison between SpykeTorch and the dedicated C++/CUDA implementations of the network proposed by Mozafari et al. (2019) and measured the training and inference time. Both networks are trained for 686 epochs (2 for the first, 4 for the second, and 680 for the last trainable layer). In each training or inference epoch, the network sees all of the training or testing samples, respectively. Note that during the training

TABLE 1 | Comparison of C++/CUDA and SpykeTorch scripts simulating the network proposed by Mozafari et al. (2019).

Script	Total training	Inference epoch	Accuracy
C++/CUDA	174,120 s (= 2d 00h 22m 20s)	35 s	97.2%
SpykeTorch	121,600 s (= 1d 09h 46m 40s)	20 s	96.9%

Both scripts are executed on a same machine with Intel(R) Xeon(R) CPU E5-2697 (2.70 GHz), 256G Memory, NVIDIA TITAN Xp GPU, PyTorch 1.1.0, and Ubuntu 16.04.

of the last trainable layer, each training epoch is followed by an inference epoch.

As shown in **Table 1**, SpykeTorch script outperformed the original implementation in both training and inference times. The small performance gap is due to some technical differences in functions' implementations and performing a new round of parameter tuning fills this gap. We believe that SpykeTorch has the potentials of even more efficient computations. For example, adding batch processing to SpykeTorch would result in a large amount of speed-up due to the minimization of CPU-GPU interactions.

6. CONCLUSIONS

In recent years, SNNs have gained many interests in AI because of their ability to work in a spatio-temporal domain as well as energy efficiency. Unlike DCNNs, most of the current SNN simulators are not efficient enough to perform large-scale AI tasks. In this paper, we proposed SpykeTorch, an open-source high-speed simulation framework based on PyTorch. The proposed framework is optimized for convolutional SNNs with at most one spike per neuron and time-to-first-spike information coding scheme. SpykeTorch provides STDP and R-STDP learning rules but other rules can be added easily.

The compatibility and integrity of SpykeTorch with PyTorch have simplified its usage specially for the deep learning communities. This integration brings almost all of the PyTorch's features functionalities to SpykeTorch such as the ability of just-in-time optimization for running on CPUs, GPUs, or Multi-GPU platforms. We agree that SpykeTorch has hard limitations on type of SNNs, however, there is always a trade-off between computational efficiency and generalization. Apart from the increase of computational efficiency, this particular type of SNNs are bio-realistic, energy-efficient, and hardware-friendly that are getting more and more popular recently.

We provided a tutorial on how to build, train, and evaluate a DCSNN for digit recognition using SpykeTorch. However, the resources are not limited to this paper and additional scripts and documentations can be found on SpykeTorch's GitHub page. We reimplemented various works (Kheradpisheh et al., 2018; Mozafari et al., 2018; Mozafari et al., 2019) by SpykeTorch and reproduced their results with negligible difference.

Although the current version of SpykeTorch is functional and provides the main modules and utilities for DCSNNs (with at most one spike per neuron), we will not stop here and our plan is to extend and improve it gradually. For example, adding automation utilities would ease programming the network's forward pass resulting a more readable and cleaner code. Due to the variations of training strategies, designing a general automation platform is challenging. Another feature that improves SpykeTorch's speed is batch processing. Enabling batch mode might be easy for operations like convolution or pooling, however, implementing batch learning algorithms that can be run with none or a few CPU-GPU interactions is hard. Finally, implementing features to support models for other modalities such as the auditory system makes SpykeTorch a multi-modal SNN framework.

DATA AVAILABILITY

The dataset analyzed for this study can be found in this link http://yann.lecun.com/exdb/mnist/.

AUTHOR CONTRIBUTIONS

MM, MG, AN-D, and TM sketched the overall structure of SpykeTorch, revised, and finalized the manuscript. MM implemented the whole SpykeTorch package and wrote the first version of the manuscript.

FUNDING

This research was partially supported by the Iranian Cognitive Sciences and Technologies Council (Grant no. 5898) and by the French Agence Nationale de la Recherche (grant: Beating Roger Federer ANR-16-CE28-0017-01).

ACKNOWLEDGMENTS

The authors would like to thank Dr. Jean-Pierre Jaffrézou for proofreading this manuscript and NVIDIA GPU Grant Program for supporting computations by providing a high-tech GPU.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). "Tensorflow: a system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA), 265–283.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a python tool for building large-scale functional brain models. *Front. Neuroinformatics* 7:48. doi: 10.3389/fninf.2013.00048

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems* (Montréal, QC), 795–805.

Bi, G. Q., and Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Brzosko, Z., Zannone, S., Schultz, W., Clopath, C., and Paulsen, O. (2017). Sequential neuromodulation of hebbian plasticity offers mechanism for effective reward-based navigation. *eLife* 6, 1–18. doi: 10.7554/eLife.27756

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Carnevale, N. T., and Hines, M. L. (2006). *The NEURON Book*. Cambridge, UK: Cambridge University Press.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Neural Networks (IJCNN), 2015 International Joint Conference on* (Killarney: IEEE), 1–8.

Falez, P., Tirilly, P., Bilasco, I. M., Devienne, P., and Boulet, P. (2019). Multi-layered spiking neural network with target timestamp threshold adaptation and stdp. *arXiv: 1904.01908*.

Ferré, P., Mamalet, F., and Thorpe, S. J. (2018). Unsupervised feature learning with winner-takes-all based stdp. *Front. Comput. Neurosci.* 12:24. doi: 10.3389/fncom.2018.00024

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Comput.* 19, 1468–1502. doi: 10.1162/neco.2007.19.6.1468

Frémaux, N., and Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Front. Neural Circuits* 9:85. doi: 10.3389/fncir.2015.00085

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature* 383:76. doi: 10.1038/383076a0

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., et al. (2018). Recent advances in convolutional neural networks. *Patt. Recogn.* 77, 354–377. doi: 10.1016/j.patcog.2017.10.013

Hazan, H., Saunders, D. J., Khan, H., Sanghavi, D. T., Siegelmann, H. T., and Kozma, R. (2018). Bindsnet: a machine learning-oriented spiking neural networks library in python. *Front. Neuroinformatics* 12:89. doi: 10.3389/fninf.2018.00089

Hussain, S., Liu, S.-C., and Basu, A. (2014). "Improved margin multi-class classification using dendritic neurons with morphological learning," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on* (Melbourne, VIC: IEEE), 2640–2643.

Kasabov, N. K. (2014). Neucube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Netw.* 52, 62–76. doi: 10.1016/j.neunet.2014.01.006

Kheradpisheh, S. R., Ganjtabesh, M., and Masquelier, T. (2016). Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing* 205, 382–392. doi: 10.1016/j.neucom.2016.04.029

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Liu, T., Liu, Z., Lin, F., Jin, Y., Quan, G., and Wen, W. (2017). "Mt-spike: a multilayer time-based spiking neuromorphic architecture with temporal error backpropagation," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (Irvine, CA), 450–457.

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e31. doi: 10.1371/journal.pcbi.0030031

Mink, J. W., Blumenschine, R. J., and Adams, D. B. (1981). Ratio of central nervous system to body metabolism in vertebrates: its constancy and functional basis. *Am. J. Physiol. Regul. Integr. Compar. Physiol.* 241, R203–R212. doi: 10.1152/ajpregu.1981.241.3.R203

Mostafa, H. (2018). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2019). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Patt. Recogn.* 94, 87–95. doi: 10.1016/j.patcog.2019.05.015

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated stdp. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 6178–6190. doi: 10.1109/TNNLS.2018.2826721

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *arXiv: 1901.09948*.

O'Connor, P., Neil, D., Liu, S. C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.* 7:178. doi: 10.3389/fnins.2013.00178

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in pytorch," in *NIPS-W* (Long Beach, CA).

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. *Neural Comput.* 29, 2352–2449. doi: 10.1162/neco_a_00990

Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (Montréal, QC), 1419–1428.

Stimberg, M., Goodman, D. F., Benichoux, V., and Brette, R. (2014). Equation-oriented specification of neural models for simulations. *Front. Neuroinformatics* 8:6. doi: 10.3389/fninf.2014.00006

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2018). Deep learning in spiking neural networks. *Neural Netw*. 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Tavanaei, A., and Maida, A. S. (2016). Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning. *arXiv: 1611.03000*.

Thiele, J. C., Bichler, O., and Dupret, A. (2018). Event-based, timescale invariant unsupervised online deep learning with stdp. *Front. Comput. Neurosci.* 12:46. doi: 10.3389/fncom.2018.00046

Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *Nature* 381:520. doi: 10.1038/381520a0

Vaila, R., Chiasson, J., and Saxena, V. (2019). Deep convolutional spiking neural networks for image classification. *arXiv: 1903.12272*.

Vitay, J., Dinkelbach, H. Ü., and Hamker, F. H. (2015). Annarchy: a code generation approach to neural simulations on parallel hardware. *Front. Neuroinformatics* 9:19. doi: 10.3389/fninf.2015.00019

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Yousefzadeh, A., Masquelier, T., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017). "Hardware implementation of convolutional stdp for on-line visual feature learning," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on* (Baltimore, MD: IEEE), 1–4.

Yu, Q., Tang, H., Tan, K. C., and Li, H. (2013). Rapid feedforward computation by temporal encoding and learning with spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 24, 1539–1552. doi: 10.1109/TNNLS.2013.2245677

Zenke, F., and Ganguli, S. (2018). Superspike: supervised learning in multilayer spiking neural networks. *Neural Comput.* 30, 1514–1541. doi: 10.1162/neco_a_01086

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 06 August 2019
doi: 10.3389/fnins.2019.00812

Unsupervised Learning on Resistive Memory Array Based Spiking Neural Networks

Yilong Guo, Huaqiang Wu, Bin Gao and He Qian*

Institute of Microelectronics, Tsinghua University, Beijing, China

Spiking Neural Networks (SNNs) offer great potential to promote both the performance and efficiency of real-world computing systems, considering the biological plausibility of SNNs. The emerging analog Resistive Random Access Memory (RRAM) devices have drawn increasing interest as potential neuromorphic hardware for implementing practical SNNs. In this article, we propose a novel training approach (called greedy training) for SNNs by diluting spike events on the temporal dimension with necessary controls on input encoding phase switching, endowing SNNs with the ability to cooperate with the inevitable conductance variations of RRAM devices. The SNNs could utilize Spike-Timing-Dependent Plasticity (STDP) as the unsupervised learning rule, and this plasticity has been observed on our one-transistor-one-resistor (1T1R) RRAM devices under voltage pulses with designed waveforms . We have also conducted handwritten digit recognition task simulations on MNIST dataset. The results show that the unsupervised SNNs trained by the proposed method could mitigate the requirement for the number of gradual levels of RRAM devices, and also have immunity to both cycle-to-cycle and device-to-device RRAM conductance variations. Unsupervised SNNs trained by the proposed methods could cooperate with real RRAM devices with non-ideal behaviors better, promising high feasibility of RRAM array based neuromorphic systems for online training.

Keywords: unsupervised learning, spiking neural network (SNN), memristor, RRAM (resistive random access memories), 1T1R RRAM, STDP

OPEN ACCESS

Edited by:
Peng Li,
University of California, Santa Barbara, United States

Reviewed by:
Amirali Amirsoleimani,
University of Toronto, Canada
Valerio Milo,
Politecnico di Milano, Italy

****Correspondence:***
Huaqiang Wu
wuhq@tsinghua.edu.cn

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *01 March 2019*
Accepted: *22 July 2019*
Published: *06 August 2019*

Citation:
Guo Y, Wu H, Gao B and Qian H (2019) Unsupervised Learning on Resistive Memory Array Based Spiking Neural Networks. Front. Neurosci. 13:812. doi: 10.3389/fnins.2019.00812

1. INTRODUCTION

Spiking Neural Networks (SNNs) have been developed in the last decades as the third generation Artificial Neural Networks (ANNs) since SNNs behave more similarly to the natural neural systems, such as the human brain (Maass, 1997). The human brain is capable of complex recognition or reasoning tasks at relatively low power consumption and in a smaller volume, compared with those of training conventional ANN models of similar accuracy. The synaptic modification manners found in cultured hippocampal neurons introduced a great abstract model of the synaptic plasticity (Bi and Poo, 1998), namely the Spike-Timing-Dependent Plasticity (STDP). The STDP rule describes how the intermediate synapse changes its plasticity according to the spike timings of pre-neurons and post-neurons. The STDP rule could be armed as an unsupervised learning mechanism in SNNs, to implement more bio-like neural computing systems. However, SNN simulations require much more effort for preserving and utilizing the enormous amount of spatial-temporal information encoded in spike trains, thus are incredibly compute-intensive on conventional von

Neumann computing systems. Some dedicated Very-Large-Scale Integration (VLSI) neuromorphic architectures have been proposed to enhance the neural simulation performance (Schemmel et al., 2010; Painkras et al., 2013; Qiao et al., 2015). VLSI technology allows intensive integration of neurons; however, the implementation of synapse arrays requires many transistors and intricate circuit designs, to emulate the learning and plasticity dynamics such as STDP. Recently, the analog Resistive Random Access Memory (RRAM) devices have become emerging neuromorphic hardware for artificial synapses, thanks to the controllability on their conductances and the ability of in-memory computing (Jo et al., 2010). The nanoscale fabricated RRAM devices can also be easily integrated as high-density crossbar arrays, which provide elegant solutions for the implementation of numerous synapses in neural systems. STDP allows the synapse to modulate its plasticity/strength according to the relative spike timing difference of the neurons connected by that synapse, and RRAM devices have been proved to be capable of providing various STDP characteristics (Jo et al., 2010; Yu et al., 2011b; Ambrogio et al., 2013, 2016; Wang et al., 2015; Pedretti et al., 2017; Wu and Saxena, 2017; Prezioso et al., 2018).

Typically, training neural network models *in-situ* on memristive devices could be challenging due to the device imperfectness and non-idealities, such as read noise, write noise, write nonlinearities, asymmetric SET/RESET switching behaviors and the limited gradual levels during programming (Agarwal et al., 2016; Chang et al., 2017; Wu et al., 2017). To accomplish recognition tasks such as learning handwritten digits (LeCun et al., 1998) with memristive neuromorphic hardware, Gokmen and Vlasov (2016) gave an estimate for the number of states that are required to be stored on a RRAM device as 600. While the reported state-of-art technologies allow the memristive devices to have 64 states (Park et al., 2016), up to over 200 states (Gao et al., 2015) continuously tuned by consecutive programming pulses, it is typically impossible to precisely control the conductance level using single shot programming (Kuzum et al., 2011; Yu et al., 2013; Eryilmaz et al., 2016). For neural networks trained with supervision, such as backpropagation (LeCun et al., 1989), the conductance of memristive devices can be fine-tuned to the desired value during the training process, using write-verification scheme (Guan et al., 2012; Yao et al., 2017), which introduces operation overheads to modulate the device conductance more precisely.

However, when it comes to unsupervised neural networks such as SNNs trained with STDP, write-verification scheme is not compatible with unsupervised learning since there is no error propagating backward and the weights should be self-adaptive to the input stimulus and output responses (STDP). Therefore, the switching behavior under consecutive programming pulses of RRAM devices is essential for implementing unsupervised learning algorithms. The dynamic range and minimum achievable mean conductance change will limit the learning rate of training algorithms (Gokmen and Vlasov, 2016). The learning rates for typical SNN training algorithms are set at the magnitude order around $10^{-4} \sim 10^{-2}$ (Masquelier and Thorpe, 2007; Querlioz et al., 2013; Panda et al., 2018), which implies at least $100 \sim 1,000$ intermediate states are needed for RRAM devices to implement such learning rules without compromise. So far, memristive device technologies could provide with devices of <100 multi-level states (Kuzum et al., 2013), which limits the complexity of RRAM-based SNNs. Several SNNs of simple structures have been simulated or demonstrated basing on memristive devices (Wang et al., 2015; Pedretti et al., 2017), accomplishing recognition tasks such as 4×4 binary patterns with one post-neuron (Pedretti et al., 2017), 3×3 binary patterns with two competitive post-neurons (Pedretti et al., 2017) and one single 8×8 pattern with eight pre-neurons and eight post-neurons (Wang et al., 2015). The abrupt switching behavior of RRAM devices limits the complexity of recognition tasks accomplished by unsupervised SNNs. Boybat et al. (2018) have proposed an architecture to wrap several Phase Change Memory (PCM) devices as one single synapse, to reduce the smallest achievable mean conductance change, therefore improving the effective conductance change granularity. This N-in-1 (N PCMs serving as one single synapse) architecture requires additional arbitration control circuit to manage N PCMs for each synapse. Their unsupervised SNN simulation with device model achieves remarkable performance by using 9-in-1 architecture (9 PCMs as one synapse), reaching testing accuracy over 70% on MNIST dataset with a single-layer (no hidden layer) SNN of 50 post-neurons, which is close to the float-precision baseline 77.2% (Boybat et al., 2018).

In this work, we propose a novel scheme for training unsupervised SNNs, with pattern/background phases and greedy training, to cooperate with realistic RRAM characteristics. The pattern/background phases and greedy training methods allow input pattern spike trains to have much lower frequencies and still guarantee the synapses to learn correct patterns and forget irrelevant information as well. Lower firing rate of neurons in SNNs will lead to fewer times of conductance changes for RRAM devices. We conduct simulations of unsupervised SNNs for the recognition of the handwritten digits from MNIST dataset, as well as the SNNs with different levels of RRAM cycle-to-cycle and device-to-device variations. The testing accuracy for 10,000 test images from MNIST dataset reaches around 75% after single-epoch unsupervised learning on 60,000 training images, with 30% cycle-to-cycle and device-to-device write variation, together with 10% cycle-to-cycle, and device-to-device dynamic range variation. The SNNs trained with proposed training methods show excellent performance even with large learning rates, which indicates that the requirement for the number of levels of RRAM devices could be reduced, and the abrupt switching, asymmetric switching could also be tolerated well. The unsupervised SNNs trained with proposed training methods show high feasibility of RRAM array based neuromorphic systems for online training.

In this article, the material details of our 1T1R device will first be introduced in section 2.1. Then the STDP architecture on 1T1R array and the unsupervised SNN architecture will be explained in sections 2.2, 2.3 respectively. The STDP characteristic of 1T1R devices measured from experiment is shown in section 3.1. The pattern/background phases and greedy training methods are described in sections 3.2.1, 3.2.2. The inference technique is also included in section 3.2.3. And classification results on digit recognition are shown in sections

3.2.4, 3.2.5. In section 4, more types of RRAM non-idealities are discussed, such as endurance, failure rate and asymmetric switching behavior. Section 5 highlights the main contributions of this work.

2. MATERIALS AND METHODS

2.1. 1T1R Device

The one-transistor-one-resistor (1T1R) structure is used to fabricate the RRAM crossbar array, as illustrated in **Figure 1**. Each RRAM device consists of a $TiN/TaO_y/HfO_x/TiN$ stack. The transistor inside the 1T1R cell plays an important role to overcome the shortcomings of the conventional 2-terminal 1R or one-selector-one-resistor (1S1R) crossbar array, such as sneak current path and programming disturbance (Yao et al., 2015). Furthermore, the gate node offers more control over the whole 1T1R cell since the current through the device can be complied during the SET processes. The control on gate voltage allows more immunity to the switching voltage magnitude and achieves better uniformity (Liu et al., 2014).

Each 1T1R cell has three main terminals: transistor gate, top electrode and transistor source, and they are connected to the word-line (WL, also noted as G), bit-line (BL), and source-line (SL) respectively in the array layout. Typical switching behavior during SET and RESET is shown in **Figure 1C**, where abrupt switching during SET is observed since the generation of each oxygen vacancy during the SET process can increase the local electric field/temperature and accelerate the generation of other vacancies, analogous to avalanche breakdown (Yao et al., 2017). Gate voltage pulses are usually different during SET and RESET processes: lower gate voltage is applied during SET to limit the set current, while RESET process requires a higher gate voltage to supply adequate reset current (Wu et al., 2011; Yao et al., 2017). Furthermore, we can notice that the switching behaviors of SET and RESET are asymmetric, which is one of the major bottlenecks that limit the performance of memristive-based neural computing system (Kuzum et al., 2013). Fortunately, this asymmetric behavior could be partly compensated by tuning device-independent parameters of proposed training methods. In the next section, we introduce an architecture for 1T1R crossbar array to implement the biological plausible STDP feature of synapses. This schematic is a general design which can be configured for different 1T1R devices that require different operating voltages.

2.2. STDP Implementation on 1T1R Array

Figure 2 shows the schematic to implement STDP characteristics basing on the 1T1R array, where each 1T1R cell acts as one electrical synapse. The pre-neuron layer is connected to the synapse array via n BLs, and the post-neuron layer is connected to m SLs, representing the fully-connected structure of two layers in topology. In the forward mode, when the pre-spike voltage signal is applied on the BL, corresponding current flows through 1T1R cell and adds up with the current of other cells in the same row at the SL node. This current stimulates the post-neuron (leaky-integrate-and-fire neuron) to integrate and modify the membrane voltage. Once the membrane voltage of the post-neuron reaches a certain threshold, the spike generator module will generate two synchronized spike signals: post-spike and gate-control. In the feedback mode, the gate line is controlled by a certain pulse generated by post-neuron, for the RRAM SET/RESET processes. The voltage across the given memristor (i, j) is determined by the voltage difference of BL_j and SL_i. So the overlapped waveform of pre-spikes and post-spikes with some time window will determine the behaviors of 1T1R cells during the feedback process. This design provides a flow paradigm with two communication phases and allows parallel modulation on crossbar states utilizing the overlapped spiking events naturally. Thanks to the crossbar architecture which binds all Gate nodes and Source nodes of all devices in one row, the temporal all-to-all spike-interaction of STDP could be implemented easily (Morrison et al., 2008). Similar structures on STDP implementation have been proposed for 1R (RRAM without any transistor, also known as 0T1R) devices (Yu et al., 2011b; Wu and Saxena, 2017; Prezioso et al., 2018), while for 1T1R devices, additional control on Gate nodes is required.

Figure 3 shows the abstract waveform design for the STDP architecture mentioned above. According to the STDP rule observed in natural neural system (Bi and Poo, 1998), when the post-spike fires slightly before the pre-spike, the synapse should be depressed, and for the RRAM device, the conductance should decrease. As illustrated in **Figure 3A**, the positive part of post-spike pulse overlaps with the negative part of the pre-spike pulse, causing a larger negative voltage across the 1T1R cell, which in fact is a RESET operation given the appropriate gate voltage, leading the synapse conductance to a lower value. Similarly in **Figure 3B**, when the post-spike follows the pre-spike closely, the voltage across the cell is a large positive value which can SET the device into a higher conductance state. **Figure 3C** shows the situation that the pre-spike does not overlap with the post-spike, and no learning mechanism is triggered. The peak positive voltage values of BL and SL are annotated as $V_{BL}{}^{+}$ and $V_{SL}{}^{+}$, and $V_{BL}{}^{-}$, $V_{SL}{}^{-}$ for the negative parts. $V_G{}^{SET}$ and $V_G{}^{RESET}$ represents the appropriate gate voltage during SET and RESET respectively. Analytically, magnitude of the voltage across the cell varies from $|V_{SL}{}^{-}|$ to $V_{BL}{}^{+} + |V_{SL}{}^{-}|$ during SET, from $V_{SL}{}^{+}$ to $V_{SL}{}^{+} + |V_{BL}{}^{-}|$ during RESET. These pulse shaping parameters (including $V_G{}^{SET}$, $V_G{}^{RESET}$ and pulse width) can be configured with flexibility to meet the control requirements of different 1T1R devices and for desired synaptic characteristics (**Figure 3D**). The STDP characteristic shown by our 1T1R devices under this scheme design is experimentally measured in section 3.1.

2.3. Unsupervised SNN Architecture

The work uses a Spiking Neural Network which consists of two layers of neurons, as shown in **Figure 4A**. The neurons in the input layer are Poisson neurons which produce spike trains whose firing rate is proportional to the associated pixel intensity (Diehl and Cook, 2015; Boybat et al., 2018). For one gray-scale image stimulus, the 2-dimensional image will be flattened into a 1D vector, and each pixel is mapped to one input Poisson neuron. The Poisson neurons are fully connected to a layer of Leaky-Integrate-and-Fire (LIF) neurons, serving as the output layer. The mechanism of one LIF neuron is explained in **Figure 4B**. In

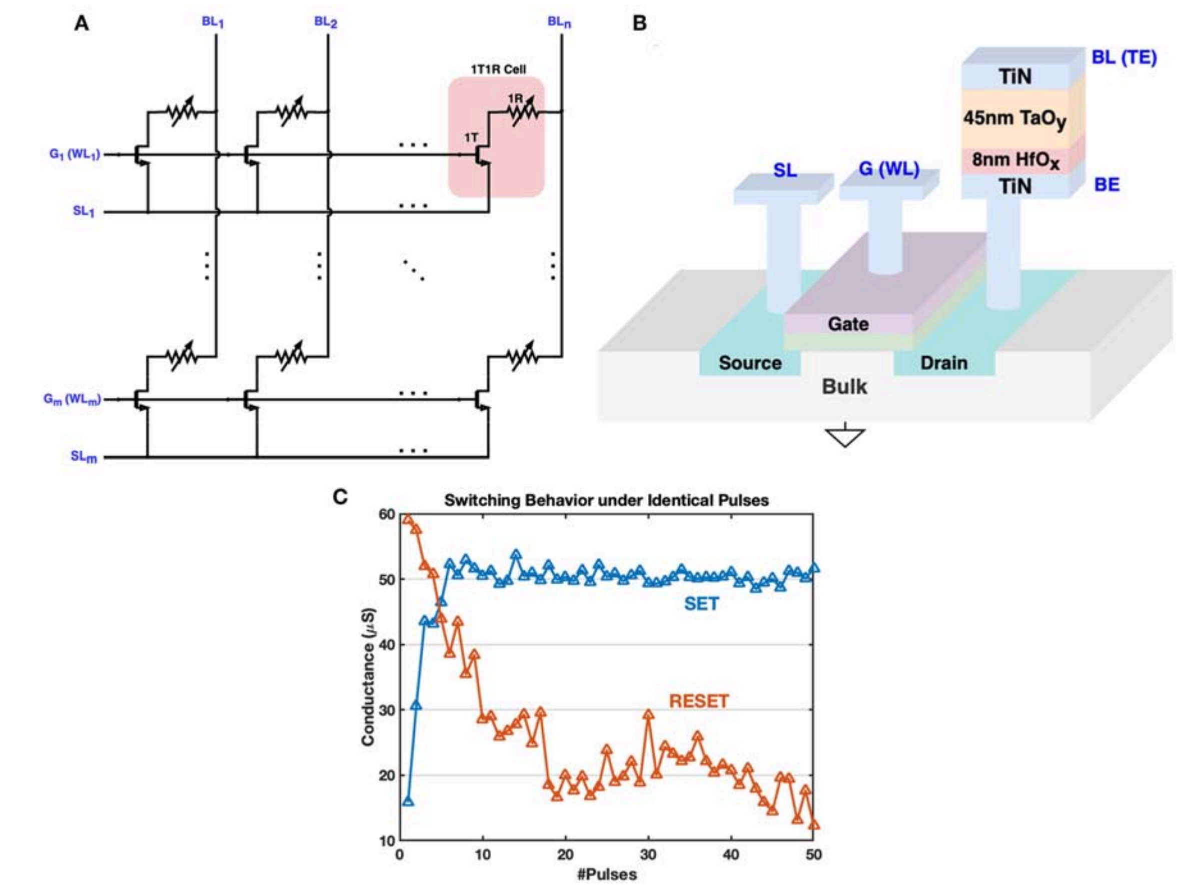


FIGURE 1 | The architecture of 1T1R crossbar array and 3D fabrication illustration. **(A)** The 1T1R crossbar array layout. The transistor gates and transistor sources of 1T1R cells in the same row are connected to the G (WL) bus and SL bus respectively. The RRAM top electrodes (TE) of 1T1R cells in the same column are gathered onto the BL bus. **(B)** The 3D fabrication structure schematic of the 1T1R cell. The bottom electrode (BE) of each RRAM device is connected to the transistor drain node., and the top electrode (TE) is wired with the BL bus. When the transistor gate is open by the high voltage on the WL bus, a positive voltage across BL and SL will help to strengthen conductive filaments in the HfO_X/TaO_y layer, increasing the RRAM conductance, which is known as the SET operation. FORM operation is similar but with a higher positive voltage across BL and SL, to form the main conductive filaments in the TaO_y layer for the first time. RESET requires a reverse operation voltage that tries to cut off the filaments formed in the HfO_X layer, thus decreasing the RRAM conductance. **(C)** Typical switching behavior of our 1T1R device under consecutive identical operation pulses (width = 50 ns) during SET/RESET. $V_{BL} = 1.5V, V_G = 2.0V, V_{SL} = 0$ for SET, and $V_{SL} = 1.4V, V_G = 4.0V, V_{BL} = 0$ for RESET. Abrupt switching is more readily observed during SET.

the forward mode, each synapse in the middle conveys the spike signals of the certain input neuron to the output neuron via its strength, defined as W. In the feedback (backward) mode, the strength of the synapse is modified according to the pre-spike and post-spike timings. The STDP variant rule, which changes weight with soft bound is used (Kistler and Hemmen, 2000), as shown in Equation 1. The relative weight changes $\Delta W/W$ of soft bound STDP model vary with different W states (see Equation 2). In general, when applying the same SET operation on RRAM devices in HRS, the consequent relative conductance change is often larger than that of devices in lower resistance states, and similarly for the RESET operation. This nonlinear manner of RRAM devices matches the synapse strength modulation modeled by soft bound STDP. The STDP model with soft bound fits better with the experimental behavior of the 1T1R device under the STDP circuit architecture and waveform design mentioned above, as explained in section 3.1. Δt is defined as $t_{\text{post}} - t_{\text{pre}}$, where t_{post} and t_{pre} represent the spike timings of the post-neuron and pre-neuron respectively. While the classical STDP model which expects the relative weight changes to be irrelevant with original weight states (see Equation 3) does not match the typical nonlinear behaviors of RRAM devices.

$$\Delta W = \begin{cases} A_+(W_{\max} - W)\exp\left(-\frac{\Delta t}{\tau_+}\right), & \text{if } \Delta t > 0 \\ -A_-(W - W_{\min})\exp\left(-\frac{|\Delta t|}{\tau_-}\right), & \text{if } \Delta t < 0 \end{cases} \quad (1)$$

$$\frac{\Delta W}{W} = \begin{cases} A_+\left(\frac{W_{\max}}{W} - 1\right)\exp\left(-\frac{\Delta t}{\tau_+}\right), & \text{if } \Delta t > 0 \\ -A_-\left(1 - \frac{W_{\min}}{W}\right)\exp\left(-\frac{|\Delta t|}{\tau_-}\right), & \text{if } \Delta t < 0 \end{cases} \quad (2)$$

$$\frac{\Delta W}{W} = \begin{cases} A_+\exp\left(-\frac{\Delta t}{\tau_+}\right), & \text{if } \Delta t > 0 \\ -A_-\exp\left(-\frac{|\Delta t|}{\tau_-}\right), & \text{if } \Delta t < 0 \end{cases} \quad (3)$$

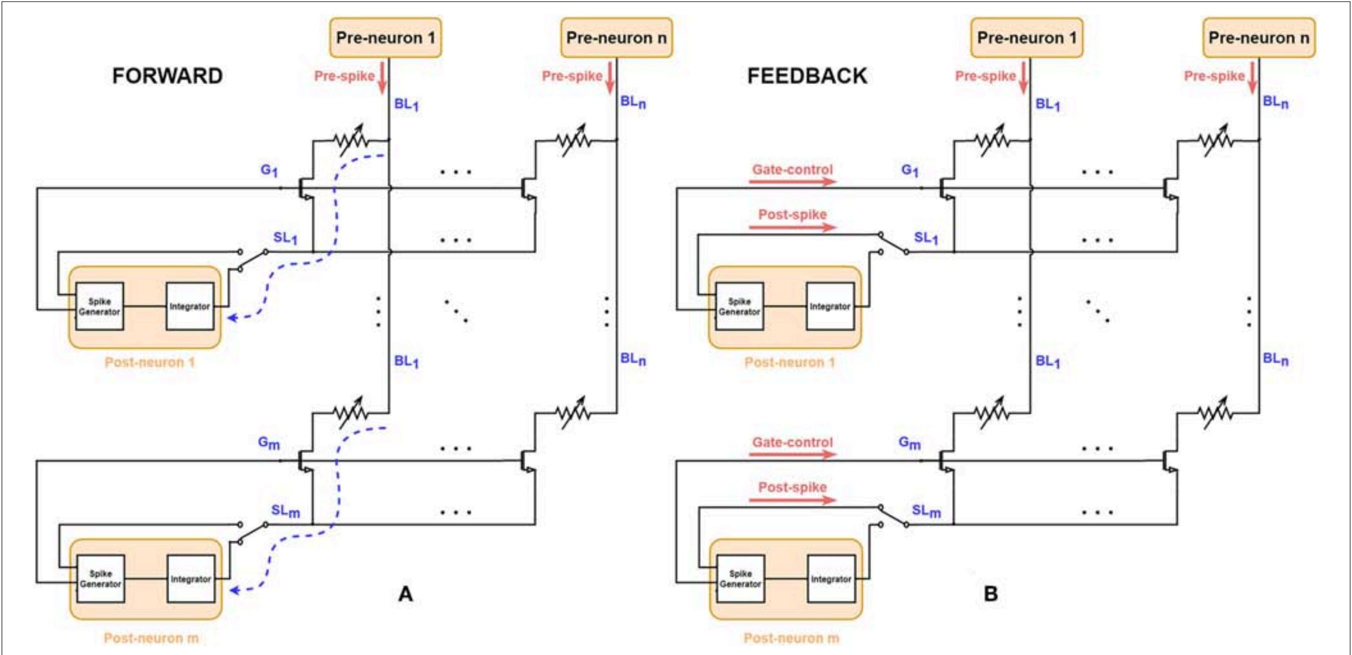


FIGURE 2 | Schematic for FORWARD/FEEDBACK modes on 1T1R RRAM array. Each Leaky-Integrate-and-Fire neuron (namely post-neuron) is connected to the SL and G nodes and each Poisson neuron (pre-neuron) is connected to the BL. **(A)** In FORWARD mode, the current stimulated by input pre-spikes can flow through the 1T1R cell and finally arrives at the integrator module of post-neurons (marked as dashed blue curve), where the input information encoded in pre-spikes is conveyed to the post-neurons. **(B)** When the post-neurons generate output signals, i.e., post-spikes and gate-controls, the circuit changes to the FEEDBACK mode via the control of the two-state switch at SLs. The conductance of RRAM devices could be programmed since the Gate is enabled and there is a voltage across the RRAM devices because of the simultaneous presence of pre-spikes and post-spikes.

Since the synapse strength is modulated by STDP rule in an unsupervised manner, competition mechanism is required for the post-neurons to learn discriminated patterns (Masquelier et al., 2009; Carlson et al., 2013; Diehl and Cook, 2015; Panda et al., 2018). Lateral inhibitory paths are added to the output neurons in Winner-Take-All (WTA) fashion: once a LIF neuron fires at t_{post}, membrane voltage of all neurons in the output layer will be reset to the resting voltage, and the spiking neuron itself goes into a refractory period as illustrated in **Figure 4B**. All other neurons need to re-accumulate their membrane voltage from resting voltage, and the spiked one will be held at resting potential during refractory, allowing LIF neurons to compete with each other for the firing opportunity. Furthermore, the homeostasis mechanism is also introduced among LIF neurons. The membrane threshold of each LIF neuron is adapted according to its recent spiking activity: threshold of the LIF neuron with more recent firing events will increase to lower its firing opportunity during the next several stimuli, and vice versa.

The training methods, namely pattern/background phases and greedy training, which allow the SNN to cooperate with large conductance change step shown by real RRAM devices will be introduced later in section 3.2, where the performance on the MNIST recognition tasks is also discussed.

3. RESULTS

3.1. STDP Characteristic of 1T1R Device

As mentioned above, the soft bound STDP (Equation 1) models different relative weight changes of different weight states (Equation 2), and the STDP model curves of different W states are plotted in **Figure 5**. The programming pulses of designed waveforms (**Figure 3**) are applied to 1T1R devices repeatedly with different initial states using Keithley 4200A-SCS, and the conductance changes of devices are measured. **Figure 6** shows the obtained experimental data provided with detailed operation information, indicating that the designed pulse waveforms can modulate the 1T1R devices' conductance similar to the synapse behavior modeled by soft bound STDP.

The A_+, A_- parameters in Equation 1 could be regarded as the learning rate of the STDP model. For our devices, the typical fitted value of A is larger than 0.5, up to 1.0, which indicates strong potentiation and depression processes (abrupt switching shown in **Figure 1C**) of the RRAM devices. The advance in material and structure of RRAM devices will lead to more ideal behaviors, such as gradual conductance switching, linear switching and more stable intermediate conductance states, which would allow us to model the learning mechanism with smaller learning rates. In typical SNN training algorithms, the learning rates are set at the magnitude order around $10^{-4} \sim$

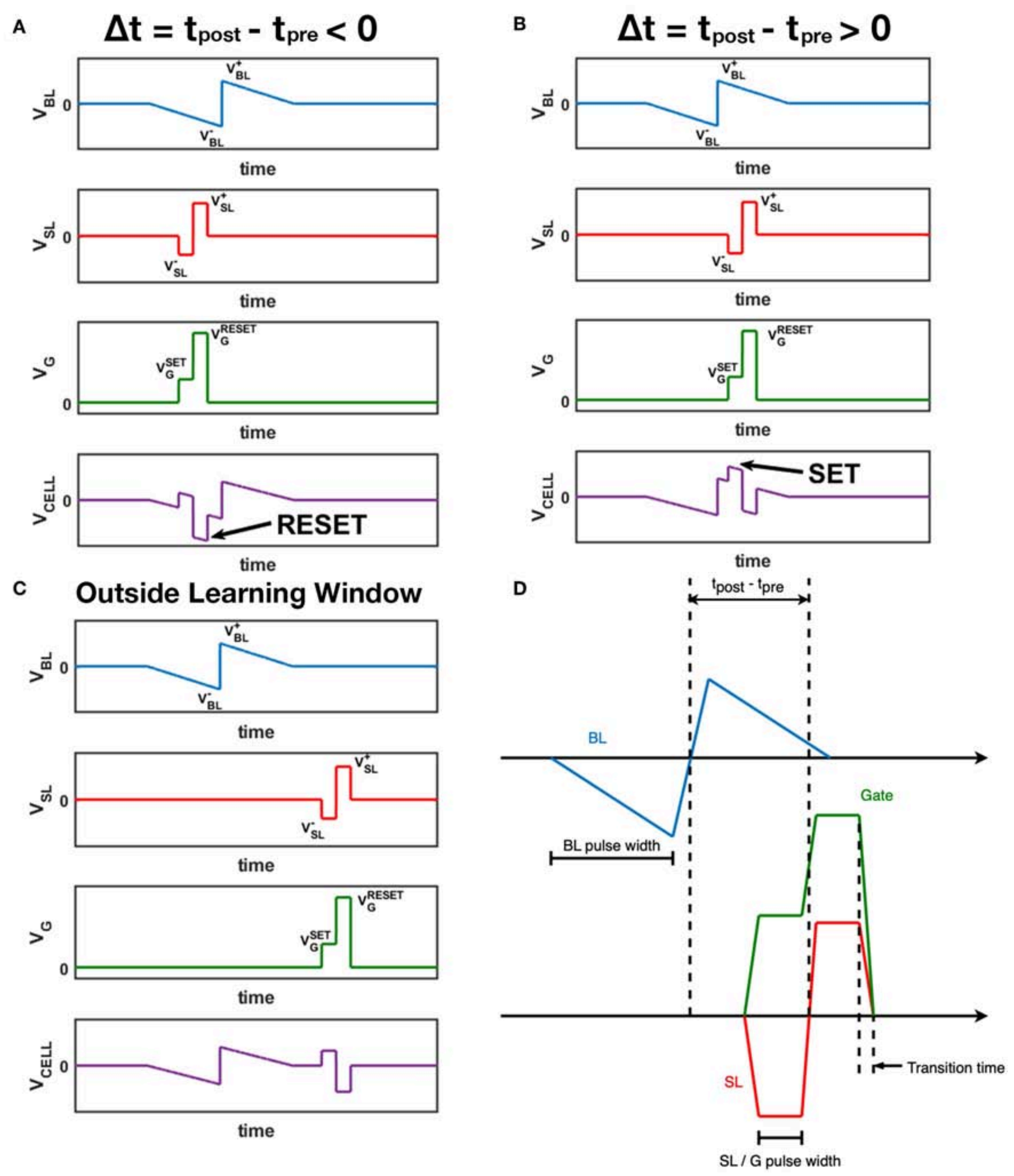


FIGURE 3 | Waveform design for BL (pre-spikes), SL (post-spikes) and Gate. The voltage across the 1T1R cell is also displayed as V_{CELL}, which equals to $V_{BL} - V_{SL}$. **(A)** A post-spike that fires right before the pre-spike event. **(B)** A post-spike that fires right after the pre-spike event. **(C)** A post-spike that fires without overlapping of the pre-spike event. **(D)** Time parameters of three channels. The transition time of all channels are the same, and SL pulses and Gate pulses have the same synchronized width.

10^{-2} (Masquelier and Thorpe, 2007; Querlioz et al., 2013; Panda et al., 2018), which would face immense difficulties applying on current general RRAM devices directly without other circuit aids. To cooperate with the non-ideal abrupt switching on RRAM conductances, we propose a novel training workflow for SNNs, named as pattern/background phases and greedy training methods (see sections 3.2.1, 3.2.2), which show immunity to large conductance changes as well as the device variations.

3.2. SNN Performance on MNIST

3.2.1. Encoding Input: Pattern/Background Phases

MNIST handwritten digits dataset is used as the application proof of SNNs trained with proposed methods. The dataset consists of

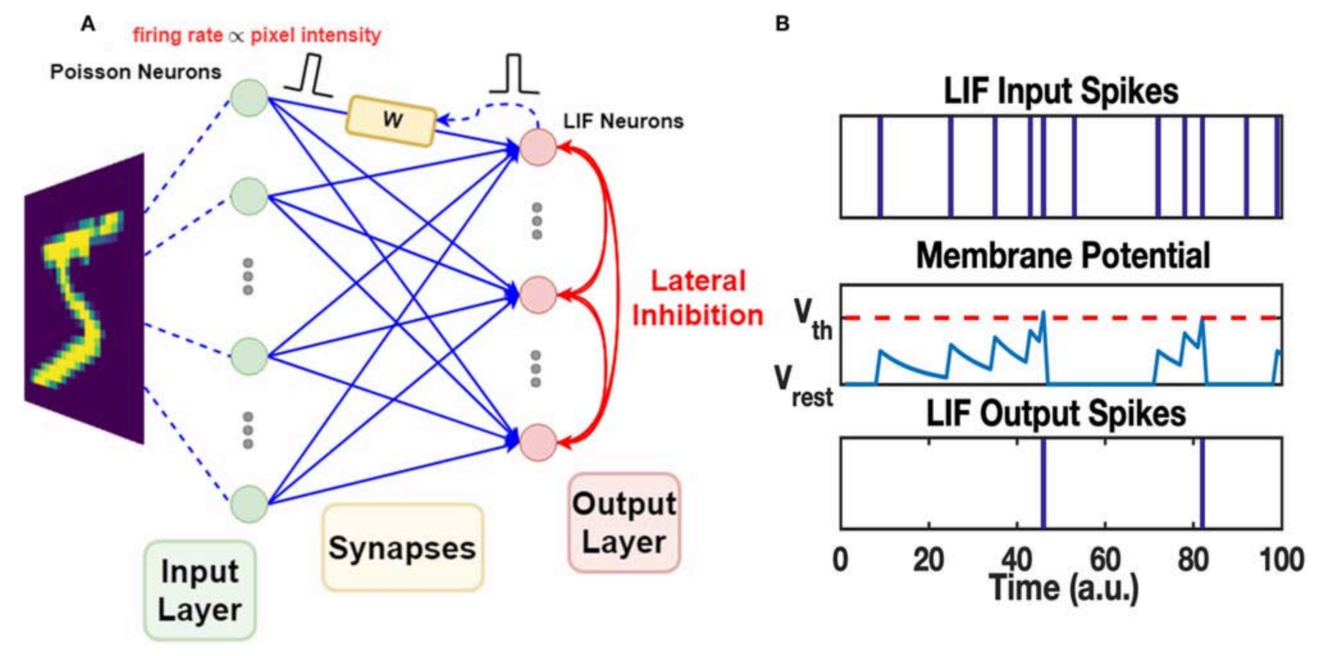


FIGURE 4 | The architecture of SNN and mechanism of LIF neuron. **(A)** The two-layer SNN architecture. The input layer is responsible for converting input images into spike trains. Poisson neurons are used in this layer. The spikes generated by the input layer are transmitted to the synapses in the middle, fully connecting the input neurons and the output neurons. The synapses modulate the received spikes (defined as pre-spikes) by their weights and pass the spikes to the output layer. LIF neurons in the output layer process the spikes and generate output spikes properly. The mechanism of LIF neuron is explained in **(B)**. The output spikes (defined as post-spikes) are passed back to the corresponding synapses and tune the synapse weights via STDP rule. Additionally, output spikes are broadcasted among output neurons through the lateral inhibition paths, allowing competition during learning. **(B)** LIF neuron firing mechanism. The LIF neuron has an internal state, i.e., membrane potential. It integrates on the presence of received input spikes and decays exponentially with a time constant τ_{mem}. Once the membrane potential reaches a certain threshold V_{th}, it fires a spike at the output port and the membrane potential is reset to the resting potential V_{rest}. The fired LIF neuron itself then enters into a short refractory period, when its membrane potential holds at V_{rest} and does not respond to any recent input spikes.

60,000 28-by-28 gray-scale images for training, and other 10,000 unseen images of the same size for testing phase[1]. Each Poisson neuron in the input layer is responsible for converting one pixel of the input image into a temporal spike train. The generated spike events are subject to Poisson distribution and firing rate of the Poisson neuron is proportional to the corresponding pixel's intensity (Diehl and Cook, 2015). At each simulation timestep, independent Bernoulli trials are conducted to determine whether to fire a spike event (Boybat et al., 2018). Additionally, the original gray-scale images from MNIST dataset are normalized by their total pixel intensity respectively before stimulating the Poisson neurons.

For each input image, the input encoding scheme includes a pattern phase and a background phase. During the pattern phase, the original image is fed to the input neurons; therefore, the pattern pixel (of higher intensity) channels are likely to have more spikes generated. During the following background phase, the complementary of the original image is used to stimulate the input layer for another period. The Poisson neurons connected to the background pixels (of lower intensity in the original image) spike more frequently in the background phase, to depress the irrelevant synapses which are mapped to the background pixels.

3.2.2. Greedy Training

The simulation is conducted at a time step of 50 ns, to match the time scale of the waveform configurations mentioned in **Figure 6**. The routine of the training process can be described as follows and shown as the block diagram in **Figure 7**:

1. Get the k-th image $I^{(k)}$ from MNIST training set.
2. Normalize $I^{(k)}$ by its total pixel intensity. Let $I_i^{(k)}$ be the intensity of the i-th pixel ($i = 1, 2, \cdots, 784$), then $\tilde{I}_i^{(k)} \leftarrow I_i^{(k)} / \sum_i I_i^{(k)}$.
3. Pattern phase: The normalized $\tilde{I}_i^{(k)}$ is mapped to the i-th Poisson neuron P_i in the input layer. For P_i, the probability to fire a spike at a given time t equals to $f_{\text{pattern}} \times \tilde{I}_i^{(k)}$, where f_{pattern} is a factor to control the overall activity of the input layer. Note that $\sum_i f_{\text{pattern}} \tilde{I}_i^{(k)} = f_{\text{pattern}}$, which represents the average number of total spiking events in the input layer at a single time step, as shown in **Figure 8A**. In this work, $f_{\text{pattern}} = 1$ is used for all simulations, so that the average firing rate of one Poisson neuron is $1/(50\,\text{ns} \times 784) \approx 25.5\,\text{kHz}$.
4. The duration of the pattern phase is variable, with a maximum of 10 μs (200 steps). $\tilde{I}^{(k)}$ is persisted to stimulate input layer

[1]The MNIST dataset used for this study can be found in THE MNIST DATABASE of handwritten digits.

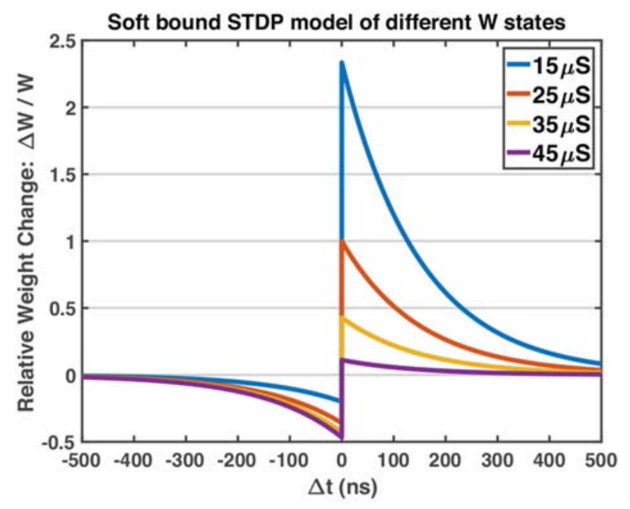


FIGURE 5 | The STDP model curves of different W states. The potentiation of lower conductance states is stronger than that of higher conductance states, and vice versa for the depression process. Model parameters: $A_+ = 1.0, A_- = 0.6, \tau_+ = \tau_- = 150\,\text{ns}, W_{\text{max}} = 50\,\mu\text{S}, W_{\text{min}} = 10\,\mu\text{S}$. A_- is set to be smaller than A_+, which fits the experimental behavior of RRAM devices in **Figure 6**.

until one post-neuron finally reaches its membrane threshold and fires a post-spike. Then pattern phase is switched to background phase immediately. The input layer expects to activate only one post-neuron during the pattern phase, this is so-called "greedy" training (**Figure 8B**).

5. Background phase: The complementary version of $I^{(k)}$ is defined as $\bar{I}^{(k)} = 255 - I^{(k)}$. Normalization is also conducted to the complementary image, such that normalized $\tilde{\bar{I}}_i^{(k)} \leftarrow \bar{I}_i^{(k)} / \sum_i \bar{I}_i^{(k)}$. Similarly, the normalized complementary image stimulates the input layer by a factor $f_{\text{background}} = 7$, resulting in an average firing rate of one Poisson neuron at around 128 kHz, as illustrated in **Figure 8C**. The background phase has a constant duration of 500 ns (10 steps).
6. Training iteration process of image $I^{(k)}$ is completed. Get the $(k+1)$-th image from MNIST training set. Repeat from step 2 to step 6.

For LIF neurons in the output layer, the membrane time constant $\tau_{\text{mem}} = 10\,\mu\text{s}$. Resting membrane potential $V_{\text{rest}} = 0\text{V}$, and initial firing threshold is set as $V_{\text{th}} = 0.4\text{V}$. The refractory period is disabled for simplicity. Winner-Take-All rule is used for lateral inhibition, that is, only one LIF neuron in the same layer is allowed to fire in any single time step (Masquelier et al., 2009). Once some neuron fires a spike, membrane potentials of all neurons in that layer are reset to V_{rest}. If more than one neuron's membrane potential increases over the firing threshold in one simulation time step, the one that exceeds its threshold the most is fired. The threshold of each LIF neuron is adapted through homeostasis: it increases by $0.1 \times (A - T)$ at every new image input, where A represents the average number of spikes per time step for recent 1,000 images' training iterations, and T is the target number of spikes per time step (Boybat et al., 2018).

For synapses which fully connect the input and output layers, the soft bound model defined by Equation 1 is used. The parameters fitted with device experimental behaviors are used: $A_+ = 1.0, A_- = 0.6, \tau_+ = \tau_- = 150\,\text{ns}, W_{\text{max}} = 50\,\mu\text{S}, W_{\text{min}} = 10\,\mu\text{S}$. Initial synapse weights are uniformly distributed in $[W_{\text{min}}, W_{\text{max}}]$.

3.2.3. Inference Process

After iterating over all training images for one time, the network will be set to static inference mode. The synapse weights and membrane thresholds of LIF neurons will remain unchanged during the inference process. The lateral inhibition mechanism is still enabled to allow competition among output neurons, and the greedy manner is also kept, therefore once some post-neuron fires a spike for the input stimulus, the inference for this input is completed. The training images are applied to the network once again, and each image is persisted to stimulate the network until some post-neuron fires. The fired neuron index and firing time are recorded. Each image with label gives the fired neuron a confidence score as $\frac{1}{\text{firing time}}$ for the corresponding label, which indicates that the earlier the output neuron fires, the more confident the neuron is. The scores are summed up for each neuron and label after the stimulation of all training images, and all the LIF neurons are marked with the label with the highest summed confidence score. Then for any input image, once some post-neuron fires, the label corresponded with that neuron is recorded as the predicted label, which could be compared to the truth label. Therefore the recognition accuracy could be evaluated.

3.2.4. Performance Without Variations

First of all, a single pattern learning task is conducted by using proposed greedy training method (pattern/background phases technique is always included for greedy training in this article unless explicitly pointed out) and conventional training method respectively. The conventional training method is armed with self-decaying techniques to forget irrelevant information more rapidly (Panda et al., 2018). The target pattern is the first image of MNIST, a handwritten digit "5." The network consists of 784 input neurons and one single output neuron. All parameters for both training methods keep the same, except for some unique method-specific parameters such as background firing rate for greedy training and decay factor for conventional training. The efficacy of synapses is compared with the target pattern after learning since there is no supervision and competition among output neurons, and an ideal learning method should be able to learn all the details of the pattern. Therefore, the error rates of pattern pixels and background pixels are calculated to evaluate the learning accuracy, as shown in **Figure 9**. The network is trained by both methods under different learning rates, and **Figures 9A,B** show that the proposed greedy training has a better convergence especially when the learning rate is larger, and the speed for both methods is comparable (see green curves). Moreover, greedy training is also able to depress the irrelevant background synapses with the same speed as the self-decaying mechanism (Panda et al., 2018), shown in **Figures 9C,D**. The

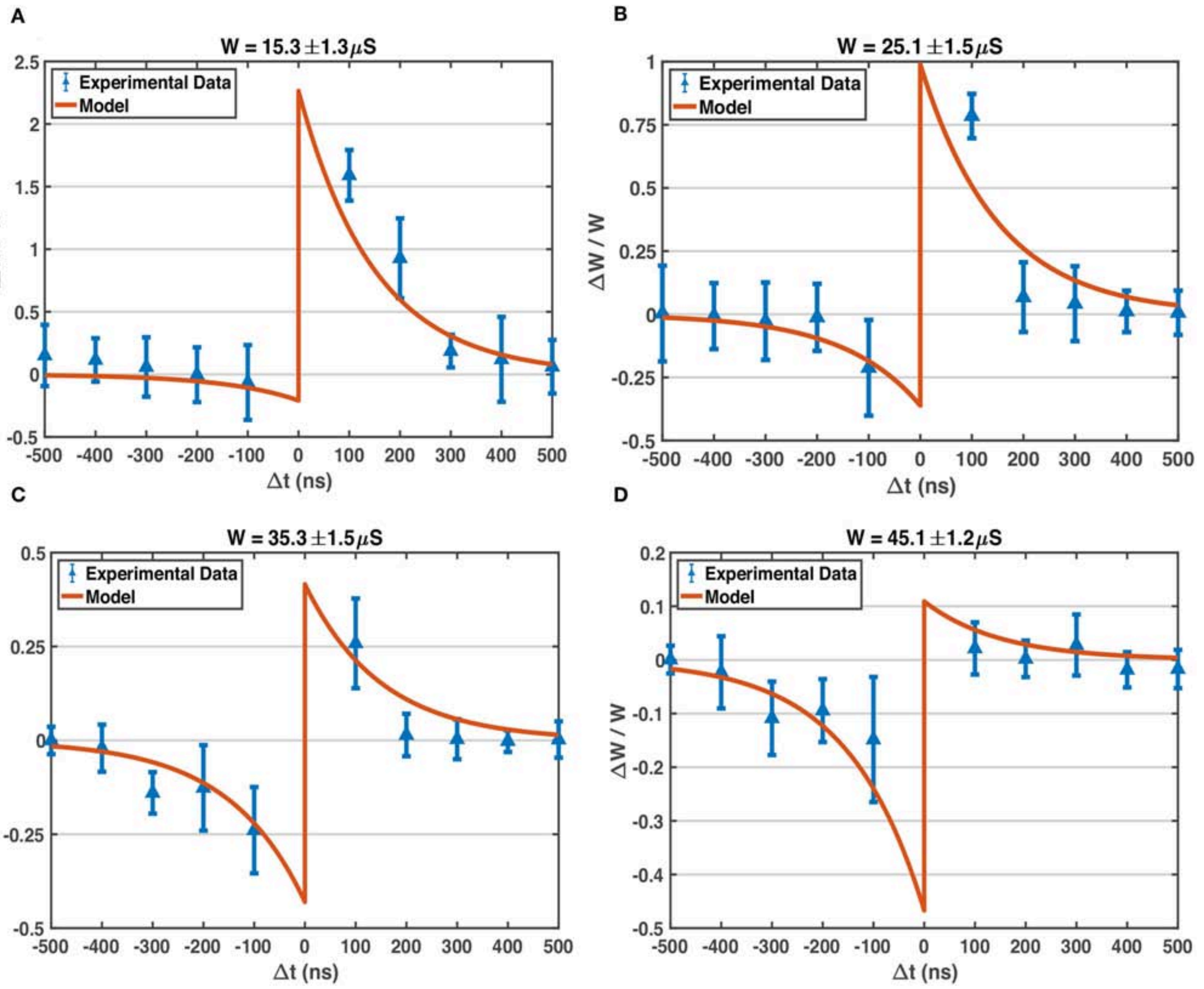


FIGURE 6 | Experimentally measured STDP characteristic of our 1T1R devices, compared with the model. The waveform parameters of BL, SL, G pulses applied on the devices: $V_{BL}^+ = 0.6$ V, $V_{BL}^- = -1.0$ V, $V_{SL}^+ = 1.3$ V, $V_{SL}^- = -1.0$ V, $V_G^{RESET} = 4.0$ V, $V_G^{SET} = 1.0$ V, pulse width of $V_{BL} = 500$ ns, pulse width of V_{SL} and $V_G = 50$ ns, all transition time = 20 ns. The model parameters used here to compare with experimental data are the same as those listed in **Figure 5**: $A_+ = 1.0, A_- = 0.6, \tau_+ = \tau_- = 150$ ns, $W_{\max} = 50\,\mu$S, $W_{\min} = 10\,\mu$S. **(A)** The experimentally measured data on 1T1R devices (blue points with errorbar) via Keithley 4200A-SCS equipment, and model-predicted STDP curve, around *W* state of 15.3 μS. Each plotted experimental data point is the average relative conductance change of over 100 trials, and the standard deviation is shown by the corresponding errorbar. In each trial, the device under test is fine-tuned to the target conductance state first, and then pulses are applied to device terminals for once, finally the conductance change is measured. **(B)** Measured STDP and modeled STDP around *W* state of 25.1 μS. **(C)** Measured STDP and modeled STDP around *W* state of 35.3 μS. **(D)** Measured STDP and modeled STDP around *W* state of 45.1 μS.

proposed training method lowers the requirement for the device characteristics, at least in terms of the minimal achievable conductance change.

We have also trained an SNN with 784 input neurons and 50 output neurons to learn and recognize the full MNIST dataset. The network is of the same structure as the one in Boybat et al. (2018) but is trained by the proposed greedy method. The parameter values are set to be device compatible as mentioned in the caption of **Figure 6** and section 3.2.2: timestep = 50 ns and $A_+ = 1.0, A_- = 0.6, \tau_+ = \tau_- = 150$ ns, $W_{\max} = 50\,\mu$S, $W_{\min} = 10\,\mu$S, $f_{\text{pattern}} = 1, f_{\text{background}} = 7$. The learning window width for STDP rule is set as four timesteps to reduce the number of update operations. The pattern phase of each training image is persisted for 200 time steps at most (since the greedy algorithm may finish the learning of this image ahead of time), and the background phase lasts for ten timesteps. Sixty thousand images from the MNIST training set are fed to the network sequentially (dataset order is not changed), and each image is learned only once. The training process finishes after around 9.6 million timesteps, which indicates that the average learning time for one image is around 160 steps, showing that greedy learning could cut ∼25% off the expected training time (210 steps for one image). The overall testing accuracy on 10,000 unseen images from MNIST testing set reaches 78.9% and is 76.8 ± 0.8% on

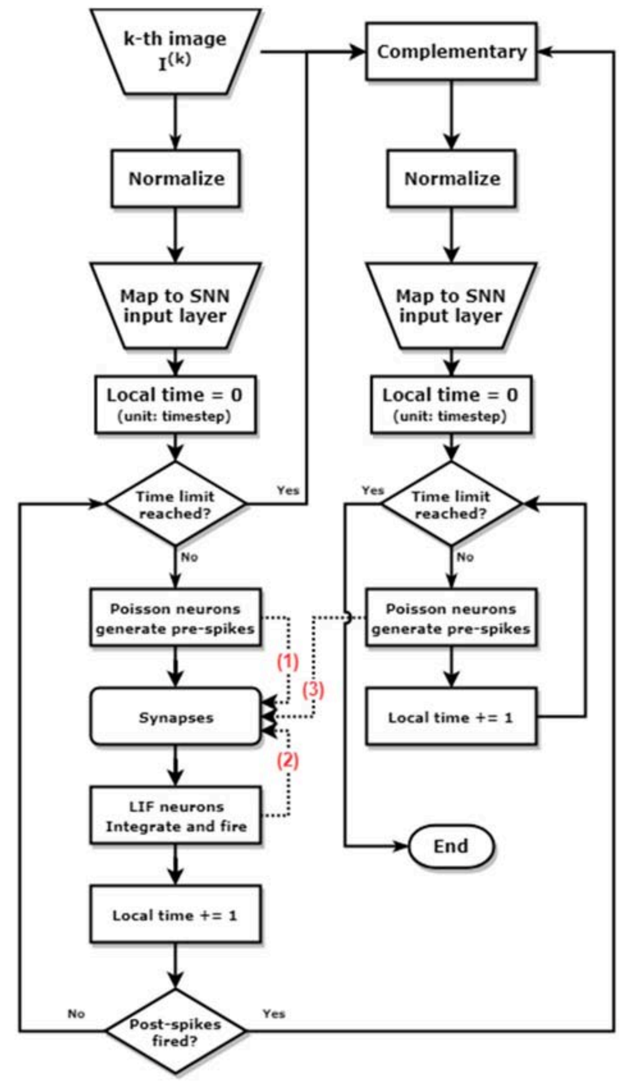


FIGURE 7 | Block diagram of pattern/background phases and greedy training methods for learning one single image. The left and the right column represent the pattern phase and background phase respectively. The dashed lines annotated with number show the different timing when spikes arrive at the synapses. The pre-spikes in the pattern phase (1) arrive first, then the post-spikes from LIF neurons (2), and finally the pre-spikes generated by the Poisson neurons in the background phase (3). This spikes arriving sequence allows potentiation during pattern phases [synapses modulated by (1) and (2)], and depression during background phases [synapses modulated by (2) and (3)].

average, as illustrated in **Figure 10**, which is comparable with the float-precision baseline of 77.2% accuracy in Boybat et al. (2018).

In the next subsection, the immunity to RRAM device variations of so-trained SNNs is explored.

3.2.5. Performance With Variations

The variations in RRAM crossbar arrays could be classified as two types: the cycle-to-cycle variation and the device-to-device variation. The cycle-to-cycle variation is mainly caused by the intrinsic stochastic physics mechanisms of the memristive devices. As mentioned in section 2.1, the conductance of our memristive devices is controlled by the states of the internal filaments. When a SET operation voltage is applied to the device, the oxygen vacancies will generate stochastically and vice versa for the RESET process. Therefore, the switching behaviors of memristive devices may vary from cycle to cycle, showing fluctuations even under the same operation conditions, which is known as the cycle-to-cycle variation. There also exists the device-to-device variation when it comes to RRAM arrays. The fabrication mismatches, line resistances, and capacitances will lead to different behaviors from device to device. For example, when pre-spikes/post-spikes are applied to one column/row of the array as illustrated in **Figure 2**, the actual voltage across each cell may vary due to the IR drop, and on the other hand, the threshold of each RRAM device is also different because of fabrication mismatches. Besides, the non-idealities of sources such as the misalignment for Gate pulses and SL pulses will also incur other variations during the training process, since the effective pulse width may vary in different operation cycles and for different cells. Proposing accurate physics and electronics models to predict the device manners is beyond the scope of this work (Yu et al., 2011a), so the impact of these variations on the proposed training methods is analyzed based on the variation of several main parameters on algorithm level, to evaluate the robustness of the proposed methods.

We have conducted repeated simulations with different levels of variation on the parameters: $A_+, A_-, W_{\max}, W_{\min}$, for both cycle-to-cycle (C2C) and device-to-device (D2D) variations. All variations are emulated by setting a certain level of the standard dispersion of the parameter, i.e., σ/μ (Querlioz et al., 2013; Agarwal et al., 2016; Gokmen and Vlasov, 2016). For D2D variation, the parameter will be sampled from the Gaussian distribution independently for all synapses before the start of one simulation, and this reference value for each synapse keep unchanged during the whole training process. If a C2C variation is also added to the simulation, the actual parameter for each synapse will be sampled from the Gaussian distribution regarding the D2D-varied value picked initially, every time the update operation happens.

The aim of the proposed greedy training method is to cooperate with the inevitable abrupt switching behavior existing in memristive devices, so the A_+, A_- parameters are set to relatively large values ($A_+ = 1.0, A_- = 0.6$ according to the experimental results in **Figure 6**), and the STDP learning window is as narrow as 4 timesteps to reduce the update operations on each synapse (update operations only happen when $|\Delta t| \leq 2\tau$). Therefore a single update may cause a ΔW at the magnitude of $8 \sim 100\%$ of the dynamic range, which indicates that 20-level devices could be sufficient for greedy training. **Table 1** shows the impact of the A_+, A_- variations. With both cycle-to-cycle and device-to-device variations, the accuracy drops from 76.8 to 73.9% at 30% variation level, which is already an extremely high level of variation for an electron device, but typical for research nanodevices (Querlioz et al., 2011). When the device-to-device A_+, A_- variation reaches 50%, around 5% of devices could not be programmed properly in at least one direction (A_+ or A_- becomes negative), i.e., , the conductance of these defected devices always decreases whenever potentiation process happens

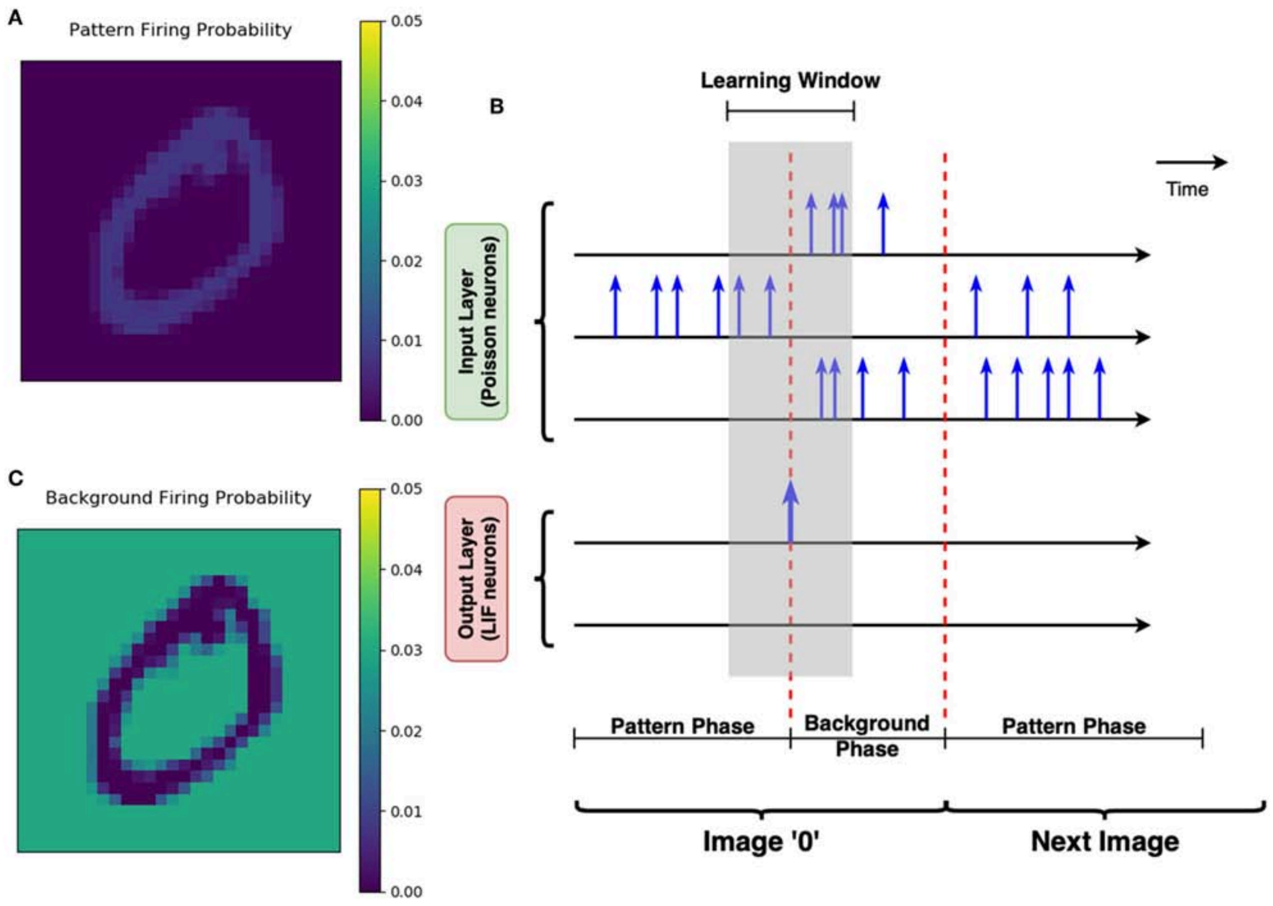

A
Pattern Firing Probability
0.05
0.04
0.03
0.02
0.01
0.00
C
Background Firing Probability
B
Learning Window
Time
Input Layer (Poisson neurons)
Output Layer (LIF neurons)
Pattern Phase
Background Phase
Pattern Phase
Image '0'
Next Image

FIGURE 8 | Greedy training and pattern/background phases. **(A)** The schematic of greedy training. Each row with the time axis represents the spiking activity of one neuron over time. Spikes are marked out by the blue vertical arrows. There are two phases for each image stimuli. During the pattern phase, the input neuron corresponding to the pattern pixels are more likely to fire spikes. The pattern phase continues until any of the LIF neurons at the output layer spikes. The LIF neuron's spike switches the input layer into the background phase immediately, allowing the background pixels to strongly stimulate the Poisson neurons. Therefore, the learning window shown by the gray shaded area could help the synapses learn the pattern and forget the irrelevant background efficiently. **(B)** The firing probability map of Poisson neurons in one single time step. This subfigure shows that the neurons corresponding to the "0" pattern have a higher firing probability during the pattern phase. **(C)** The firing probability of all pixels during the background phase. The Poisson neurons associated with background pixels are strongly stimulated during the background phase, to enable efficient forgetting of the network.

and vice versa. In this situation, the accuracy drops around 10%. However, the functionality of the network is not challenged. On the other hand, the greedy training is immune to large cycle-to-cycle write variation up to 50%, since each device may suffer from a potentiation/depression disorder with a probability of only 5%, every time the update operation happens.

We also simulated the impact of the dynamic range ($W_{\max}, W_{\min}$) variations, as shown in **Table 2**. The initial dynamic range is set to $10 \sim 50\,\mu$S, meaning that the on/off ratio equals to only 5, which is easy to fulfill for typical memristive devices (Kuzum et al., 2013). The network can tolerate 10% variation level of $W_{\max}$ and $W_{\min}$ with <2% accuracy loss, and still functions well with 30% cycle-to-cycle and device-to-device $W_{\max}, W_{\min}$ variation with a 67% testing accuracy. When the variation goes to 50%, around 10% of devices in the simulation are stuck at the initial value since the maximal conductance becomes less than minimal conductance, which incurs severe accuracy loss for MNIST application. Querlioz et al. (2011) have shown that this type of unsupervised SNN can tolerate 50% $W_{\max}, W_{\min}$ variation well, however with a dynamic range of 10^4, which allows larger variations but is hard to implement for most nanodevices.

Table 3 compares the performance between greedy-trained unsupervised SNNs and conventional-trained unsupervised SNNs (Querlioz et al., 2011; Boybat et al., 2018). The listed three networks are of the same structure, 784 inputs together with 50 output neurons. The learning increments and decrements (normalized by dynamic range) for greedy training and conventional training are compared, and we can see that conventional training requires the synapses to be able to tune their conductances at the magnitude of 0.5% to 1% regarding the switching window width ($W_{\max} - W_{\min}$), which needs

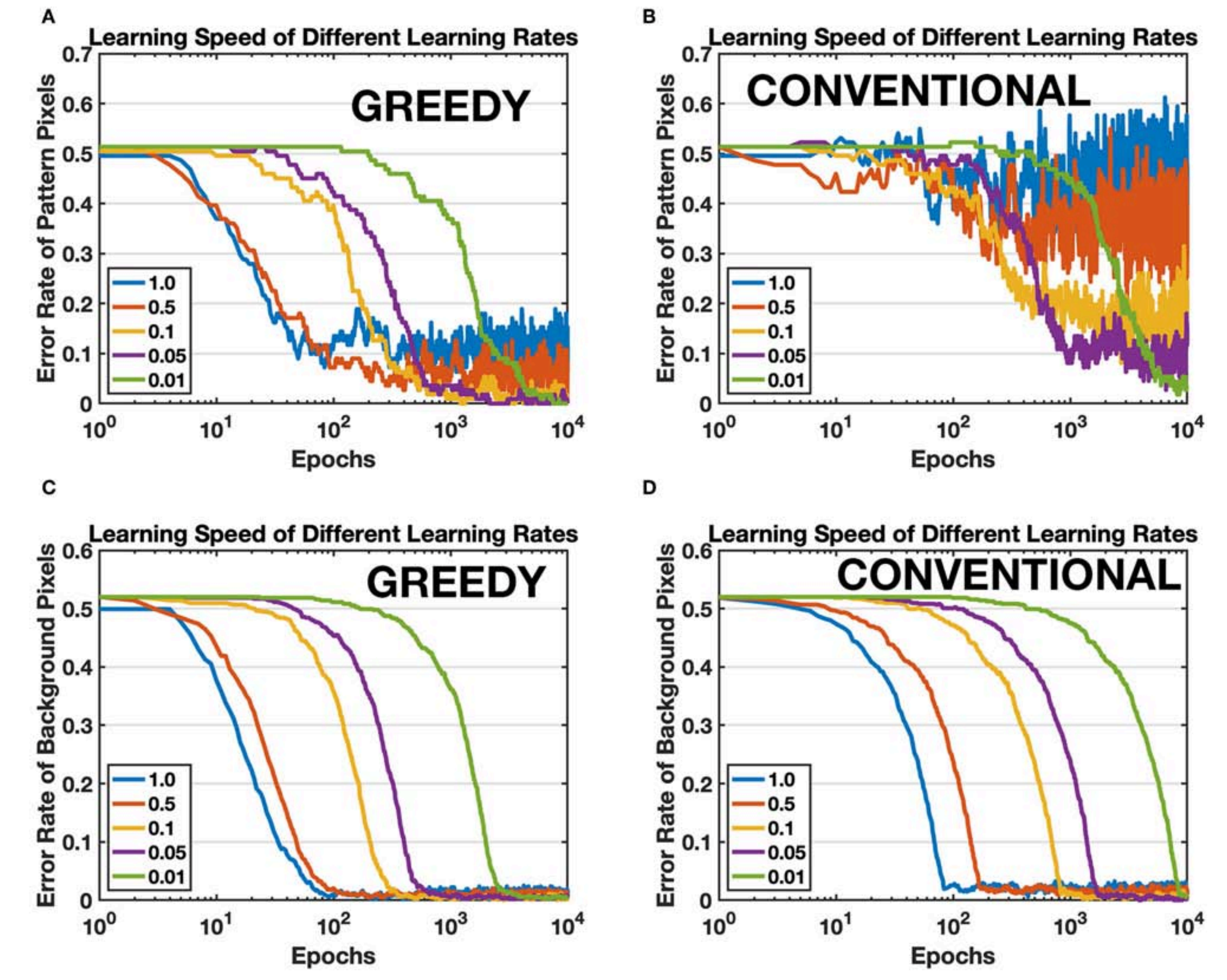


FIGURE 9 | Comparison between the proposed training method (GREEDY) and conventional training method (CONVENTIONAL) on learning one single pattern. **(A)** Error rate of pattern pixels versus training epochs of greedy training. **(B)** Error rate of pattern pixels versus training epochs of conventional training. The convergence of conventional training with large learning rates is much worse than that of greedy training. **(C)** Error rate of background pixels versus training epochs of greedy training. **(D)** Error rate of background pixels versus training epochs of conventional training.

devices to have over 200 levels under consecutive programming pulses (Querlioz et al., 2011). Since this requirement is hard to fulfill for most memristive devices (Gao et al., 2015; Park et al., 2016), an architecture wrapping N devices as one single synapse has been proposed by Boybat et al. (2018), and they have proved that training SNNs using up to 9 devices/synapse can achieve over 70% testing accuracy on MNIST, reducing the required device levels to around 20, which is easy to implement. On the other hand, the greedy training method proposed in this work dilutes the spiking activities in the time domain, and forces the synapses to learn greedily, with large learning increments and decrements of 30 to 50% regarding the switching window, therefore using one memristive device with 20 levels as one synapse could be sufficient to achieve the same functionality.

4. DISCUSSION

4.1. Device Endurance

Online training for neural networks on RRAM devices often requires a large number of conductance tuning operations, where we must consider the device endurance problem. The core concept of greedy training is to dilute spike trains in the time domain, thus reducing the number of device operations. Typical update count map after training with 60,000 images is shown in **Figure 11A**, where update count of an individual synapse is no

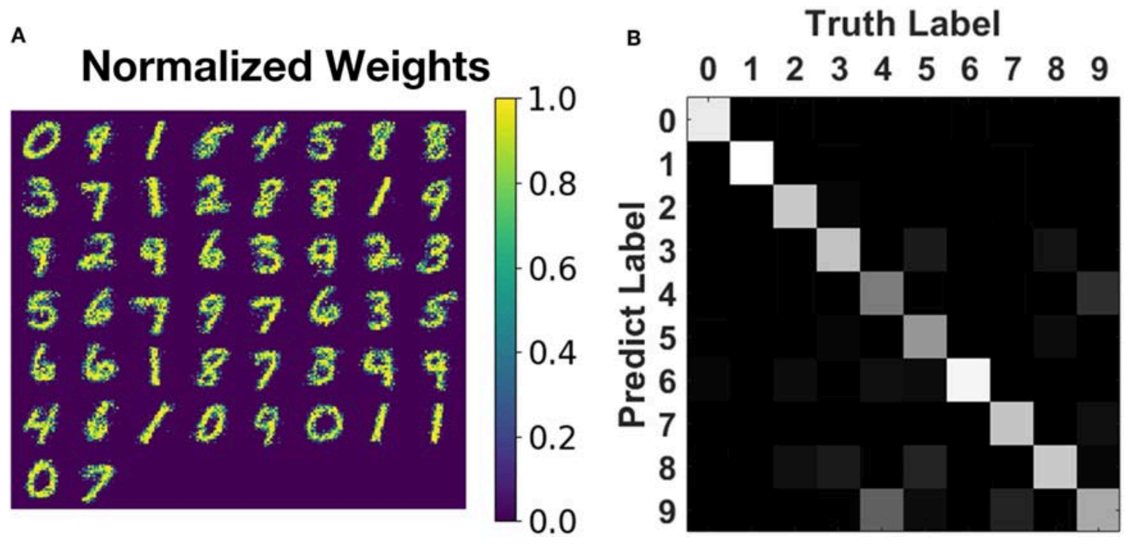


FIGURE 10 | Training result on MNIST recognition. **(A)** The normalized weight map corresponding to 50 post-neurons. Most patterns of 10 digits are impressively learned without any supervision. **(B)** Testing accuracy on MNIST testing set of 10,000 unseen images during training. The overall testing accuracy is around 76.8%, and most of the categories could be classified with acceptable accuracy.

TABLE 1 | The testing accuracy for different levels of variation on A_+, A_-.

Variation level	10%	30%	50%
C2C	76.22 ± 0.81%	75.30 ± 1.14%	74.84 ± 1.22%
D2D	76.42 ± 1.78%	76.17 ± 1.03%	65.42 ± 1.88%
Combined	75.74 ± 1.24%	73.94 ± 1.53%	63.48 ± 2.09%

Results for cycle-to-cycle (C2C) variation, device-to-device (D2D) variation, and C2C-D2D combined variation are listed. Simulations are repeated for 12 times with each condition, and the testing accuracy is shown as $\mu \pm \sigma$, where μ, σ represents the mean value and standard deviation respectively.

TABLE 2 | The testing accuracy for different levels of variation on $W_{\max}, W_{\min}$.

Variation level	10%	30%	50%
C2C	76.67 ± 0.76%	71.90 ± 1.75%	63.94 ± 1.34%
D2D	74.91 ± 1.09%	71.60 ± 0.69%	65.20 ± 1.15%
Combined	75.34 ± 0.94%	67.20 ± 2.21%	56.16 ± 1.73%

Results for cycle-to-cycle (C2C) variation, device-to-device (D2D) variation, and C2C-D2D combined variation are listed. Simulations are repeated for 12 times with each condition, and the testing accuracy is shown as $\mu \pm \sigma$, where μ, σ represents the mean value and standard deviation respectively.

more than 200 times. The endurance related problems could be ignored for greedy training learning MNIST digits since these problems usually appear after 10^5 operating pulses (Zhao et al., 2018). The parameters used by Boybat et al. (2018) indicate that the learning window for STDP lasts for over 200 timesteps, and at each time step, about ten spikes (calculated according to the MNIST statistics and firing rate mentioned) are generated at the input layer. The output layer is expected to have five spikes fired for each image as well. Therefore, an estimate of update operation number would be 200 × 10 × 5 = 10 k for one training image, while the value for the proposed greedy training is around 6 × 3 × 1 ≈ 20, reducing update operations by a factor of 500. The conventional training method may be affected by endurance related problems more severely. Besides, reducing the number of update operations could also make the algorithm more energy efficient theoretically.

TABLE 3 | Comparison table of memristive-device-based SNNs for MNIST handwritten recognition.

	This work	Boybat et al., 2018	Querlioz et al., 2011
Training method	Greedy	Conventional	Conventional
Network structure	784 × 50	784 × 50	784 × 50
Accuracy with variations	∼75%	∼70%	∼80%
Devices per synapse	1	≥9	1
Learning increments, decrements	∼0.5, ∼0.3	0.01, 0.006	0.01, 0.005
Required device levels	∼20	∼20	>200

The accuracy with variations of this work is obtained with 30% cycle-to-cycle and device-to-device A_+, A_- variation, and 10% cycle-to-cyle and device-to-device W_{max}, W_{min} variation. For Boybat et al. (2018), the N-in-1 architecture (non-differential) with N=9 and with device variation model is listed. And for Querlioz et al. (2011), the data is obtained with 25% cycle-to-cycle A_+, A_- variation, and 25% cycle-to-cycle W_{max}, W_{min} variation.

4.2. Array Failure Rate

Although the endurance related device failure problem could be ignored for greedy training, we have conducted simple simulations to explore the influence of yield. A SNN with four output neurons is used to recognize 1,000 “0,” “1” digit images, and trained with different array failure rates. The failed devices are stuck to their initial states and do not respond to any

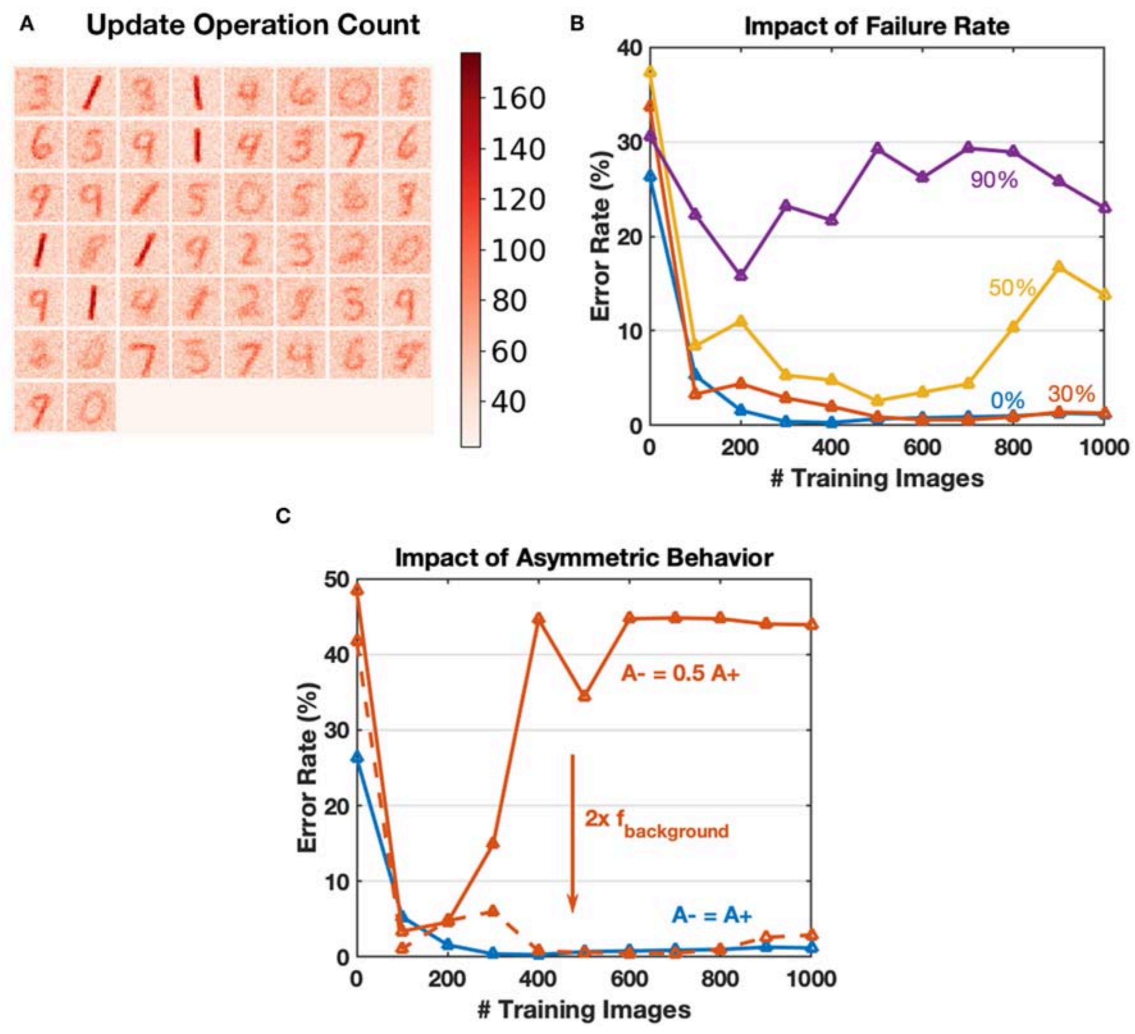


FIGURE 11 | (A) Heat map of the update counts for each synapse after learning 60,000 MNIST training images. The maximum is <200, which could be ignored for endurance related problems. **(B)** Impact of array failure rate. A simple SNN of 4 output neurons to recognize 1,000 "0," "1" images is simulated. The failed devices are kept at the initial state and do not respond to any input. The failure rate could have an impact on the convergence, and 30% failure rate can be tolerated in this application. **(C)** Using algorithm-level parameters to compensate for the common asymmetric switching behaviors for RRAM devices. For the solid red curve, double the background firing rate factor leads to similar performance with balanced switching conditions.

input during training. **Figure 11B** shows that the convergence is affected severely, especially when the failure rate goes over 50%. Since endurance issues are ignored, a typical failure rate of a functional array should be around 10% (Wu et al., 2017), and greedy training is robust for this situation.

4.3. Compensate Asymmetric Switching Behavior

Commonly, memristive devices have asymmetric switching behaviors (Kuzum et al., 2013), which is one of the bottlenecks for hardware neural networks. Thanks to the pattern/background phases of greedy training, the potentiation and depression during SNN training happen in different time slots, and the input firing rate for each phase could be configured independently. Therefore, we can compensate for the asymmetric switching behavior partly by tuning the pattern/background firing factors, as shown in **Figure 11C**.

4.4. Divide Spikes Into Pattern/Background Parts

For greedy training, it is guaranteed that potentiation happens in the pattern phase and depression in the background phase. So we can divide the pre-spikes and post-spikes into minor parts from their timing middle points, then we get a negative/positive pulse pair for each spike (the same manipulation should be applied to gate-control signals as well). The original design of waveforms in **Figure 3** requires post-spikes and gate-control signals to be synchronized well, so if the circuit non-idealities result in the misalignment of post-spikes and gate-control signals, there may cause unsafe device operations (V_G^{RESET} applied to the gate node when SET is expected). Fortunately, breaking each spike signal into two parts and operates separately in the pattern and background phase could solve this problem. Jittering between G and SL signals will only lead to a lower effective overlapped pulse width, will not cause unsafe operations anymore.

5. CONCLUSION

To work with the inevitable large conductance change step introduced by RRAM devices, we propose novel approaches of pattern/background phases and greedy training for unsupervised SNNs. Pattern/background phases and greedy training method provide an efficient workflow of unsupervised SNN learning because they make sure that only the pattern spikes occur just before the post-spike events, and background spikes will follow the post-spikes. Furthermore, greedy training guarantees that only one post-spike will be fired for each stimulus, which allows larger weight changes. The simulated SNN model manages to cooperate with the large learning rate incurred by RRAM devices by diluting spikes in the temporal dimension and therefore achieves gradual learning with very few spikes, which significantly reduce the requirement on the number of gradual levels of memristive devices from over 200 to around 20, and then could be fulfilled by typical memristive devices. The greedy-trained unsupervised SNNs also have good immunity to the conductance change variation and switching window variation and reach $\sim$75% testing accuracy on the MNIST test set with moderate variations. Furthermore, the low-density interaction fashion of greedy training reduces the number of SET/RESET operations on memristive devices by around 2 orders, for example a maximum of 200 operations is observed for single-epoch learning 60,000 MNIST training images, and this could substantially mitigate the endurance related problems which is one of the bottlenecks for memristive devices based online learning systems. This work shows the potential of RRAM devices serving as neuromorphic hardware to implement practical applications with properly-trained SNNs, even with various imperfect behaviors.

DATA AVAILABILITY

The MNIST dataset used for this study can be found in THE MNIST DATABASE of handwritten digits.

AUTHOR CONTRIBUTIONS

The ideas and methods are proposed and discussed by YG, HW, and BG. The experiments and simulations mentioned in this work are completed by YG. During the whole progress, HW, BG, and HQ all offered suggestions which help YG to carry out the research reported by this article.

FUNDING

This work was supported in part by the National Key R&D Program of China (2017YFB0405604), NSFC (61851404, 61874169, 61674089), Beijing Municipal Science and Technology Project (Z181100003218001), Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips (ICFC).

REFERENCES

Agarwal, S., Plimpton, S. J., Hughart, D. R., Hsia, A. H., Richter, I., Cox, J. A., et al. (2016). "Resistive memory device requirements for a neural algorithm accelerator," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 929–938. doi: 10.1109/IJCNN.2016.7727298

Ambrogio, S., Balatti, S., Milo, V., Carboni, R., Wang, Z., Calderoni, A., et al. (2016). "Novel rram-enabled 1t1r synapse capable of low-power stdp via burst-mode communication and real-time unsupervised machine learning," in *2016 IEEE Symposium on VLSI Technology* (Honolulu, HI: IEEE), 1–2. doi: 10.1109/VLSIT.2016.7573432

Ambrogio, S., Balatti, S., Nardi, F., Facchinetti, S., and Ielmini, D. (2013). Spike-timing dependent plasticity in a transistor-selected resistive switching memory. *Nanotechnology* 24:384012. doi: 10.1088/0957-4484/24/38/384012

Bi, G.-q., and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.

Boybat, I., Le Gallo, M., Nandakumar, S. R., Moraitis, T., Parnell, T., Tuma, T., et al. (2018). Neuromorphic computing with multi-memristive synapses. *Nat. Commun.* 9:2514. doi: 10.1038/s41467-018-04933-y

Carlson, K. D., Richert, M., Dutt, N., and Krichmar, J. L. (2013). "Biologically plausible models of homeostasis and stdp: stability and learning in spiking neural networks," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX: IEEE), 1–8. doi: 10.1109/IJCNN.2013.6706961

Chang, C.-C., Liu, J.-C., Shen, Y.-L., Chou, T., Chen, P.-C., Wang, I.-T., et al. (2017). "Challenges and opportunities toward online training acceleration using rram-based hardware neural network," in *2017 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA: IEEE), 11–6. doi: 10.1109/IEDM.2017.8268373

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Eryilmaz, S. B., Joshi, S., Neftci, E., Wan, W., Cauwenberghs, G., and Wong, H.-S. P. (2016). "Neuromorphic architectures with electronic synapses," in *2016 17th International Symposium on Quality Electronic Design (ISQED)* (Santa Clara, CA: IEEE), 118–123. doi: 10.1109/ISQED.2016.7479186

Gao, L., Wang, I.-T., Chen, P.-Y., Vrudhula, S., Seo, J.-s., Cao, Y., et al. (2015). Fully parallel write/read in resistive synaptic array for accelerating on-chip learning. *Nanotechnology* 26:455204. doi: 10.1088/0957-4484/26/45/455204

Gokmen, T., and Vlasov, Y. (2016). Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Front. Neurosci.* 10:333. doi: 10.3389/fnins.2016.00333

Guan, X., Yu, S., and Wong, H.-S. P. (2012). On the switching parameter variation of metal-oxide rrampart i: Physical modeling and simulation methodology. *IEEE Trans. Elect. Dev.* 59, 1172–1182. doi: 10.1109/TED.2012.2184545

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., and Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Lett.* 10, 1297–1301. doi: 10.1021/nl904092h

Kistler, W. M., and Hemmen, J. L. v. (2000). Modeling synaptic plasticity in conjunction with the timing of pre-and postsynaptic action potentials. *Neural Comput.* 12, 385–405. doi: 10.1162/089976600300015844

Kuzum, D., Jeyasingh, R. G., Lee, B., and Wong, H.-S. (2011). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* 12, 2179–2186. doi: 10.1021/nl201040y

Kuzum, D., Yu, S., and Wong, H. S. (2013). Synaptic electronics: materials, devices and applications. *Nanotechnology* 24:382001. doi: 10.1088/0957-4484/24/38/382001

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. et al. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324.

Liu, H., Lv, H., Yang, B., Xu, X., Liu, R., Liu, Q., et al. (2014). Uniformity improvement in 1t1r rram with gate voltage ramp programming. *IEEE Elect. Dev. Lett.* 35, 1224–1226. doi: 10.1109/LED.2014.2364171

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671.

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive stdp-based spike pattern learning. *Neural Comput.* 21, 1259–1276. doi: 10.1162/neco.2008.06-08-804

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e31. doi: 10.1371/journal.pcbi.0030031

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybernet.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Painkras, E., Plana, L. A., Garside, J., Temple, S., Galluppi, F., Patterson, C., et al. (2013). Spinnaker: a 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE J. Solid State Circ.* 48, 1943–1953. doi: 10.1109/JSSC.2013.2259038

Panda, P., Allred, J. M., Ramanathan, S., and Roy, K. (2018). Asp: learning to forget with adaptive synaptic plasticity in spiking neural networks. *IEEE J. Emerg. Select. Top. Circ. Syst.* 8, 51–64. doi: 10.1109/JETCAS.2017.2769684

Park, J., Kwak, M., Moon, K., Woo, J., Lee, D., and Hwang, H. (2016). Tio x-based rram synapse with 64-levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing. *IEEE Elect. Dev. Lett.* 37, 1559–1562. doi: 10.1109/LED.2016.2622716

Pedretti, G., Milo, V., Ambrogio, S., Carboni, R., Bianchi, S., Calderoni, A., et al. (2017). Memristive neural network for on-line learning and tracking with brain-inspired spike timing dependent plasticity. *Sci. Rep.* 7:5288. doi: 10.1038/s41598-017-05480-0

Prezioso, M., Mahmoodi, M. R., Bayat, F. M., Nili, H., Kim, H., Vincent, A., et al. (2018). Spike-timing-dependent plasticity learning of coincidence detection with passively integrated memristive circuits. *Nat. Commun.* 9:5311. doi: 10.1038/s41467-018-07757-y

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Front. Neurosci.* 9:141. doi: 10.3389/fnins.2015.00141

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans. Nanotechn.* 12, 288–295. doi: 10.1109/TNANO.2013.2250995

Querlioz, D., Bichler, O., and Gamrat, C. (2011). "Simulation of a memristor-based spiking neural network immune to device variations," in *The 2011 International Joint Conference on Neural Networks* (San Jose, CA: IEEE), 1775–1781. doi: 10.1109/IJCNN.2011.6033439

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE international symposium on Circuits and systems (ISCAS)* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Wang, Z., Ambrogio, S., Balatti, S., and Ielmini, D. (2015). A 2-transistor/1-resistor artificial synapse capable of communication and stochastic learning in neuromorphic systems. *Front. Neurosci.* 8:438. doi: 10.3389/fnins.2014.00438

Wu, H., Yao, P., Gao, B., Wu, W., Zhang, Q., Zhang, W., et al. (2017). "Device and circuit optimization of rram for neuromorphic computing," in *2017 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA: IEEE), 11–5. doi: 10.1109/IEDM.2017.8268372

Wu, M.-C., Lin, Y.-W., Jang, W.-Y., Lin, C.-H., and Tseng, T.-Y. (2011). Low-power and highly reliable multilevel operation in zro2 1t1r rram. *IEEE Elect. Dev. Lett.* 32, 1026–1028. doi: 10.1109/LED.2011.2157454

Wu, X., and Saxena, V. (2017). "Enabling bio-plausible multi-level stdp using cmos neurons with dendrites and bistable rrams," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 3522–3526. doi: 10.1109/IJCNN.2017.7966299

Yao, P., Wu, H., Gao, B., Eryilmaz, S. B., Huang, X., Zhang, W., et al. (2017). Face classification using electronic synapses. *Nat. Commun.* 8:15199. doi: 10.1038/ncomms15199

Yao, P., Wu, H., Gao, B., Zhang, G., and Qian, H. (2015). "The effect of variation on neuromorphic network based on 1t1r memristor array," in *2015 15th Non-Volatile Memory Technology Symposium (NVMTS)* (Beijing: IEEE), 1–3. doi: 10.1109/NVMTS.2015.7457492

Yu, S., Gao, B., Fang, Z., Yu, H., Kang, J., and Wong, H.-S. (2013). A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation. *Adv. Mater.* 25, 1774–1779. doi: 10.1002/adma.201203680

Yu, S., Guan, X., and Wong, H.-S. P. (2011a). "On the stochastic nature of resistive switching in metal oxide rram: Physical modeling, monte carlo simulation, and experimental characterization," in *2011 International Electron Devices Meeting* (Washington, DC: IEEE), 17–3. doi: 10.1109/IEDM.2011.6131572

Yu, S., Wu, Y., Jeyasingh, R., Kuzum, D., and Wong, H.-S. P. (2011b). An electronic synapse device based on metal oxide resistive switching memory for neuromorphic computation. *IEEE Trans. Elect. Dev.* 58, 2729–2737. doi: 10.1109/TED.2011.2147791

Zhao, M., Wu, H., Gao, B., Sun, X., Liu, Y., Yao, P., et al. (2018). Characterizing endurance degradation of incremental switching in analog rram for neuromorphic systems. in *2018 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA: IEEE), 20–2. doi: 10.1109/IEDM.2018.8614664

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 13 August 2019
doi: 10.3389/fnins.2019.00855

A Swarm Optimization Solver Based on Ferroelectric Spiking Neural Networks

Yan Fang[1]*, *Zheng Wang*[1], *Jorge Gomez*[2], *Suman Datta*[2], *Asif I. Khan*[1] *and Arijit Raychowdhury*[1]*

[1] *School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, United States,* [2] *Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN, United States*

As computational models inspired by the biological neural system, spiking neural networks (SNN) continue to demonstrate great potential in the landscape of artificial intelligence, particularly in tasks such as recognition, inference, and learning. While SNN focuses on achieving high-level intelligence of individual creatures, Swarm Intelligence (SI) is another type of bio-inspired models that mimic the collective intelligence of biological swarms, i.e., bird flocks, fish school and ant colonies. SI algorithms provide efficient and practical solutions to many difficult optimization problems through multi-agent metaheuristic search. Bridging these two distinct subfields of artificial intelligence has the potential to harness collective behavior and learning ability of biological systems. In this work, we explore the feasibility of connecting these two models by implementing a generalized SI model on SNN. In the proposed computing paradigm, we use SNNs to represent agents in the swarm and encode problem solutions with the spike firing rate and with spike timing. The coupled neurons communicate and modulate each other's action potentials through event-driven spikes and synchronize their dynamics around the states of optimal solutions. We demonstrate that such an SI-SNN model is capable of efficiently solving optimization problems, such as parameter optimization of continuous functions and a ubiquitous combinatorial optimization problem, namely, the traveling salesman problem with near-optimal solutions. Furthermore, we demonstrate an efficient implementation of such neural dynamics on an emerging hardware platform, namely ferroelectric field-effect transistor (FeFET) based spiking neurons. Such an emerging *in-silico* neuron is composed of a compact 1T-1FeFET structure with both excitatory and inhibitory inputs. We show that the designed neuromorphic system can serve as an optimization solver with high-performance and high energy-efficiency.

Keywords: ferroelectric FET, neuromorphic computing, spiking neural network, swarm intelligence, optimization

OPEN ACCESS

Edited by:
Peng Li,
University of California, Santa Barbara,
United States

Reviewed by:
Garrett S. Rose,
The University of Tennessee, Knoxville,
United States
Alice Cline Parker,
University of Southern California,
United States

****Correspondence:***
Yan Fang
yan.fang@gatech.edu
Arijit Raychowdhury
arijit.raychowdhury@ece.gatech.edu

Specialty section:
This article was submitted to
Neuromorphic Engineering,
a section of the journal
Frontiers in Neuroscience

Received: *07 March 2019*
Accepted: *30 July 2019*
Published: *13 August 2019*

Citation:
Fang Y, Wang Z, Gomez J, Datta S,
Khan AI and Raychowdhury A (2019)
A Swarm Optimization Solver Based
on Ferroelectric Spiking Neural
Networks. Front. Neurosci. 13:855.
doi: 10.3389/fnins.2019.00855

INTRODUCTION

Recent advances of deep learning models have initiated a resurgence of neural networks in the field of artificial intelligence (LeCun et al., 2015). Spiking Neural Network (SNN), as the third generation of neural networks, models the dynamic behavior of the biological neural system and focuses on the timing of the spikes (Maass, 1997). SNN utilizes spike timing to encode information and is capable of processing a significant amount of spatial-temporal information with a small number of neurons and spikes (Ghosh-Dastidar and Adeli, 2009; Ponulak and Kasinski, 2011). Meanwhile, neuromorphic computing hardware that implements SNN continue to gain increasing attention both in the industry and academia (Merolla et al., 2014; Davies et al., 2018). Moreover, recent progress of emerging nanotechnologies in devices and materials, such as resistive RAMs (RRAM) (Indiveri et al., 2013), spintronic devices (Romera et al., 2018) and metal-insulator transition (MIT) materials (Parihar et al., 2018), are facilitating real-time large-scale mixed-signal neuromorphic computing systems with the potential to bridge the energy efficiency gap between engineered systems and biological systems. SNN has been successfully applied in various computational tasks, such as visual recognition (Cao et al., 2015), natural language processing (Diehl et al., 2016), brain-computer interface (Kasabov, 2014), robot control (Bouganis and Shanahan, 2010). Recently, researchers have demonstrated ways to use networks of SNNs and similar neuromorphic systems to solve computationally more difficult problems. Of particular interest are optimization problems including NP-hard problem, such as constraint satisfaction problems (CSP) (Mostafa et al., 2015; Fonseca Guerra and Furber, 2017), vortex coloring problems (Parihar et al., 2017) and traveling salesman problems (TSP) (Jonke et al., 2016). These neural-inspired computing systems are designed exclusively so that the system converges at problem solutions by harvesting both deterministic as well as stochastic dynamics. Nonetheless, there are very few previous works about SNN based computing systems that address generic optimization problems. Although solving CSP with SNN is promising, it is enticing to note that the computational platform that we empirically find in the human brain can also solve complex optimization problems.

On the other hand, swarms of creatures also show collective behavior and evolve with complex and highly optimized global strategies. For example, a colony of ants is capable of planning the shortest path between their nest and their food sources, which is attributed to the collaborative deposit of chemical pheromone on the trails (Goss et al., 1989). A school of sardine naturally optimizes the movement of the swarm to minimize the loss when it is attacked by sharks (Norris and Schilt, 1988). Bees can build hives with an optimized structure in spatial efficiency and locate nearest nectar source plants with temporal efficiency (Michener, 1969). These swarms are composed of individuals that have inferior intelligence and simple behaviors. However, they exhibit highly intelligent collective behavior resulting from the collaboration. Inspired from these natural swarms, Swarm Intelligence (SI) constructs the computational models that describe the collaborative behaviors in decentralized and self-organized systems (Blum and Li, 2008). In recent years, SI is also applied to a wide range of fields, such as path planning, control of robotics, image processing, and communication networks (Duan and Luo, 2015). Examples of classic SI optimization methods include ant colony optimization (ACO) (Dorigo and Di Caro, 1999), particle swarm optimization (PSO) (Kennedy and Eberhart, 1999). More advanced SI optimization algorithms that have been proposed recently include the firefly algorithm (FA) (Fister et al., 2013) and bat algorithm (Yang, 2010).

SNN and SI are apparently two computational intelligence models that differ in concepts, architectures and applications. SNN is inspired by the neural system of a high-intelligent individual, while SI mimics the collaborative behavior of somewhat simpler creatures. However, these two sets of models share some similarities. Both of them are bio-inspired, highly parallelized, and composed of multiple homogeneous units (agents and neurons) (Fang and Dickerson, 2017). Their computational capabilities origin from the interaction and communication between the individual units. For example, both of the neurons in SNN and agents in SI exhibit the behavior of phase and frequency synchronization. From the perspective of computational neuroscience, synchronization of oscillatory neural activity is currently one of the attractive areas of research, due to its close connection to the rhythms of the brain, seizures in epileptic patients and tremor in Parkinson patients (Guevara Erra et al., 2017). Neural synchronization has also been utilized in neuromorphic computing based on spiking or oscillatory neural networks, such as visual processing (Fang et al., 2014), olfactory processing (Brody and Hopfield, 2003), and solving constraint satisfaction problems (Parihar et al., 2017). In these applications, neural synchronization usually indicates the completeness of computing and the stable state of dynamical systems that presents the results. Similarly, an SI model can be viewed as a discrete dynamical system with an energy function that matches the objective function of the optimization problem. Agents perform collaborative searches and eventually synchronize and cluster around the global energy minima, which represents the global optimal (or near-optimal) solution. Such synchronization phenomena in SNN and SI model are the primary inspiration of our work.

As the problem dimension and the swam sizes increase, SI algorithms can become computationally expensive in terms of delay and power. On the other hand, SNNs cannot harness the collective properties of optimization problems. In our previous work (Fang and Dickerson, 2017), we explored the opportunities in bridging these two models and proposed a computing paradigm based on SI and coupled spiking oscillator network to address optimization problems. In this work, we provide details and develop an SI-SNN architecture and demonstrate how it is capable of solving two types of optimization problems, parameter optimization of continuous objective functions and TSP.

Along with algorithm development, the next generation of computing systems must harness the computational advantages of emerging post-silicon technologies. In particular, for neuromorphic systems, research has started in earnest to identify materials and device systems that exhibit the inherent dynamics of bio-inspired neurons and synapses. Various competing technologies are being explored, including insulator-metal-transition devices (Parihar et al., 2017), RRAMs (Ielmini, 2018), spintronic neurons and synapses (Romera et al., 2018) as well as scaled silicon CMOS implementations (Indiveri and Horiuchi, 2011). In this paper, we explore the use of ferroelectric field-effect transistor (FeFET) based spiking neurons in the design of the proposed SI-SNN architecture. An algorithm-hardware co-design is required to provide the next breakthrough in computational efficiency, in particularly for neuro-inspired systems whose dynamics can be simulated, albeit inefficiently in a von-Neumann system. The FeFET based spiking neuron is a compact 1T-1FeFET *in-silico* neuron with both excitatory and inhibitory inputs (Wang et al., 2017). It takes advantage of the hysteresis of the FeFET and operates as a relaxation oscillator that periodically generates voltage spikes. We extract a simplified model to capture the critical voltages and spike timing of FeFET based spiking neuron. This compact model enables the simulation of SNN that contains a large number of neurons.

First, we show how the proposed SI-SNN organizes multiple SNNs and performs parallel meta-heuristic searching, which is conducted by a swarm of collaborative agents in an SI-inspired algorithm. In this design, the spiking neurons encode the parameters of the agents with the spiking rate, interact with each other via spikes and search for globally optimal solutions. The agents that find better solutions modulate the firing rates of neurons in other agents. The modulation behavior is performed through event-based synaptic connections. Specifically, the excitatory input voltage of a post-synaptic FeFET neuron is modulated by a small amount whenever a spike arrives. Eventually, the optimal solution is represented by the firing rates when the entire swarm synchronizes.

In the second problem demonstration, we use a similar SI-SNN computing architecture to imitate the ACO (Dorigo and Di Caro, 1999) algorithm and show how it is capable of solving the TSP. Each SNN is a winner-takes-all (WTA) network and the order of its neurons' spikes represents the traveled route (solution candidate) of a single agent (ant). The synaptic weight is updated online by the spikes and shared by multiple SNNs, resembling the pheromone trails in ACO. The travel routes of SNNs are adapted according to the distances between cities and the pheromone distribution. Consequently, the optimal solution eventually evolves though such a parallel search process.

The remaining sections of this paper are organized as follow. In Materials and Methods, we describe the dynamical behavior model of FeFET spiking neuron as a hardware platform; it is the neuron model we use to develop the SI-SNN computing paradigm. Then we introduce two SI-SNN paradigms and demonstrate solutions to different optimization problems—continuous objective functions and TSP. In section Results, we provide the simulation results of our proposed method. In the final section, we draw conclusions.

MATERIALS AND METHODS

Neuromorphic Hardware Technology

Owing to the continuous dynamics of the biological nervous systems biomimetic SNNs are much less efficient when they are executed on digital computing machines. Neuromorphic hardware that specifically supports SNN has been explored theoretically and experimentally for three decades (Mead, 1989). Nowadays neuromorphic engineering focuses on developing large-scale neural processing systems for cognitive tasks (Indiveri et al., 2011). In this work, we demonstrated a co-design of the proposed SI-SNN computing paradigm and neuromorphic hardware, where the hardware natively implements the required neuronal dynamics. A neuromorphic hardware system, comprises of two fundamental functional units:

(a) *Neuron*: This is the primary focus of this paper. Here, we explore the spiking dynamics of a FeFET neuron based on its excitatory and inhibitory interfaces and utilize this dynamical behavior to enable different SNN functionalities. The FeFET neuron has also been proven to be energy-efficient. It costs about one-third of power as traditional CMOS circuits and can potentially achieve the energy efficiency of 0.36 nJ/spike with 45 nm FinFET process (Wang et al., 2018). We discuss the detail dynamical behavior of FeFET spiking neuron in the next section.

(b) *Synapse*: Various resistive memory technologies are currently being investigated to realize synaptic behavior. The synapse does not show complex dynamics, but rather allows summation of the outputs of multiple pre-synaptic neurons to modulate the membrane potential of the post-synaptic neuron. For the sake of brevity, we do not include a detailed discussion about the hardware implementation of synapses because many emerging device technologies can fulfill the requirements of SI-SNN systems (Kuzum et al., 2013).

Ferroelectric Based Spiking Neuron

FeFET is a semiconductor device that has a similar structure as the MOSFET or FinFET, except that an additional layer of ferroelectric (FE) material is integrated into the stack of gate terminal (Aziz et al., 2018). The spontaneous polarization of the FE layer is reversible under a certain electric field applied in the correct direction. The polarization depends on the current electric field and its history, resulted in a hysteresis loop. For further details, interested readers are pointed to Aziz et al. (2018). Such a feature of FE layer induces a FeFET to switch "on" at a high voltage and "off" at a low applied gate voltage. **Figure 1** illustrate the structure of a FeFET (red box). A relaxation oscillator based on FeFET was recently proposed in Wang et al. (2017). Furthermore, the proposed oscillator was utilized to implement a spiking neuron with excitatory and inhibitory interfaces (Wang et al., 2018). The proposed circuits employ the hysteresis of a FeFET and a traditional NMOS transistor to periodically charge and discharge a load capacitor and generate spikes of voltage (**Figures 1**, **2A**). **Figure S1** shows a 3D view of the FeFET and the NMOS transistor.

The FeFET based neuron has only two transistors and exhibits an advantage in the energy efficiency of spikes, which is discussed later in section Results. More importantly, this neuron model is capable of modeling multiple neural dynamics that has been observed in cortical and thalamic neurons. We can use two gate voltages, V_{GM} and V_{GF}, of two transistors to imitate the excitatory and inhibitory synaptic inputs, respectively of biological neurons, and thus enable various neural firing patterns (Fang et al., 2019). In this section, we describe a compact behavior model of the FeFET based spiking neuron. This model captures the critical switching voltages of FeFET and computes the current that controls spike timing (phase) and spiking frequency. It neglects the complex physical transitions before device switching and reduces the computing cost tremendously, enabling the simulation of large scale SNN built on FeFET neuron.

Figure 1 depicts the schematic of a FeFET spiking neuron (Wang et al., 2017). It is a relaxation oscillator that charges and discharges the load capacitor repetitively with I_D and I_M, which are the currents flowing through the FeFET and the NMOSFET. The former one injects current to capacitor C and the latter one provides a discharging path. To briefly explain the oscillation, we assume V_{GF}, V_{GM}, and V_{DD} are all fixed. If we start from the charging phase, the potential across the capacitor, V_S, is low and thus the V_{GS} of FeFET is large enough to set the FE layer to coercion and inject charge into the gate node V_g and quickly switches on the FeFET. As a result, I_D increases rapidly and charges the capacitor until the end of this phase. As the capacitor gets charged and V_S rises, the discharging phase begins. The FE layer reaches the opposite coercive threshold, drains the charge from V_g and switches the FeFET to an OFF state. In this phase, I_D is very small and I_M gets a chance to discharge the capacitor. Due to the decrease of V_S again, the whole cycle repeats with these two phases. Therefore, V_S keeps swinging between the two critical voltages V_{t1} and V_{t2}. In **Figure 2A**, the blue waveform plots the trace of V_S, illustrates the Fast Spiking mode of a spiking neuron.

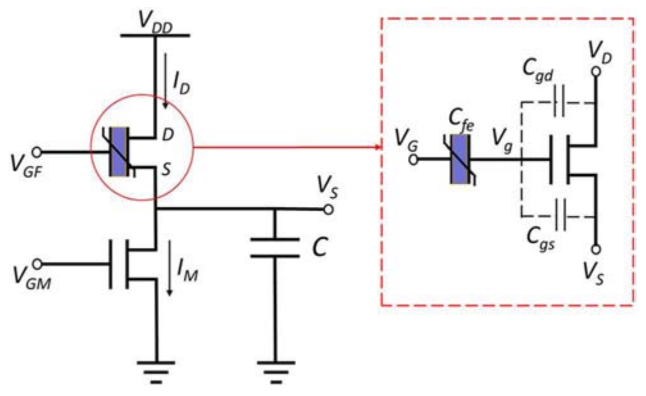


FIGURE 1 | FeFET based spiking oscillator consists of a FeFET and a normal NMOS transistor that are used to charge and discharge a capacitor. The FeFET (red box) can be view as a ferroelectric layer that connected to a common FET (3D model of FeFET is shown in **Figure S1**).

Dynamic Behavior Model

Because the switching process of FeFET is fast when compared to the oscillation period, we assume the switching of FeFET is instant in our model. We are primarily interested in the timing of the spike, instead of other physical metrics of the FeFET device. We focus our model on the critical voltages when FeFET switches and the current that charges and discharges the capacitors. Details of the model have been presented elsewhere (Fang et al., 2019) and we summarize the key findings here for the sake of completion. It is also important to point out the key neuronal dynamics that are achievable in the FeFET neuron,

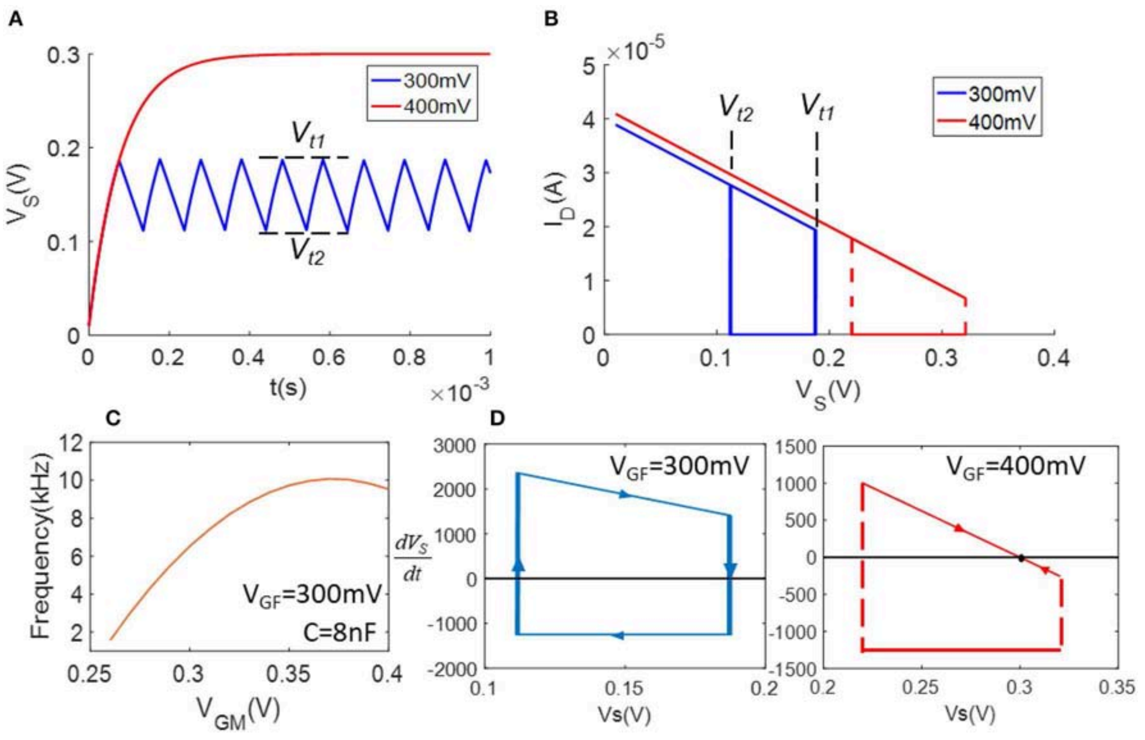


FIGURE 2 | Demonstration of model simulation: **(A)** waveforms of V_S (V_{GF} = 300 mV and 400 mV); **(B)** $I_D - V_S$ plot shows the hysteresis loops of I_D in **(A)**; **(C)** V_{GM} v.s. frequency as V_{GF} = 300 mV; **(D)** flow diagrams of equation.

that can be harnessed in the SI-SNN computational framework. Critical voltages V_{t1} and V_{t2} depend on the properties of FeFET, V_G and V_D (V_{GF} and V_{DD}) fed into the gate and drain terminals. To capture V_{t1} and V_{t2}, we only need to aim at the boundary conditions when the FeFET switches. Thus, we can write the equation based on charge (Fang et al., 2019):

$$\begin{aligned} V_g C_T &= Q_{fe} + C_{fe} V_{GF} + C_{gd} V_{DD} + C_{gs} V_S \\ C_T &= C_{fe} + C_{gd} + C_{gs} \end{aligned} \tag{1}$$

where, Q_{fe} is the released bond charge. Here $V_g = V_{GF} - V_{fe}$. V_{fe} is the potential across the FE layer and equals to one of the two coercive voltages, V_{c1} and V_{c2}. Therefore, we can compute the critical voltages of switching, V_{t1} and V_{t2} as (Fang et al., 2019):

$$\begin{aligned} V_{ti} &= \alpha^{(i)} - \gamma^{(i)} V_{DD} + \left(1 + \gamma^{(i)}\right)(V_{GF} - V_{ci}), i = 1, 2 \\ \gamma^{(i)} &= \frac{C_{gd}{}^{(i)}}{C_{gs}{}^{(i)}}, \alpha^{(i)} = -\left(\frac{C_T V_c{}^{(i)} + \beta^{(i)} Q_{fe}}{C_{gs}{}^{(i)}}\right), \beta^{(1,2)} = \pm 1 \end{aligned} \tag{2}$$

i = 1,2 represent the cases of switching on and off. $\alpha^{(i)}$, $\gamma^{(i)}$, V_{c1}, and $V_{c2}$are device parameters that can be calibrated via experimental measurements (Wang et al., 2018) or estimated from physics-based models. Thus, we can obtain V_{t1} and V_{t2} in terms of V_{GF} and V_{DD}. An alternative method to obtain V_{t1} and V_{t2} is to calibrate the data experimentally from circuits. In the case we shown here, we have ($V_{t1} = 187$ mV, $V_{t2} = 111$ mV) when V_{GF}= 300 mV ($V_{t1} = 320$ mV, $V_{t2} = 219$ mV), when V_{GF} = 400 mV.

With V_{t1} and V_{t2}, we can model the dynamical behavior of the FeFET based neuron with a first-order non-linear differential equation for V_S:

$$\begin{aligned} \frac{dV_S}{dt} &= \frac{1}{C}(sI_D - I_M), \begin{cases} s = 0, V_{t1} \rightarrow V_{t2} \\ s = 1, V_{t1} \leftarrow V_{t2} \end{cases} \\ I_D &= g_F(V_g - V_S - V_{Gth}) \\ I_M &= g_M(V_{GM} - V_{Mth}) \end{aligned} \tag{3}$$

In Equation (3), we use a binary variable s to set the current in two phases. When $s = 1$, the load capacitor is being charged, while s = 0 represent the discharging phase. I_D and I_M are modeled with two piecewise linear functions. Transistor parameters g_F, g_M, V_{Gth}, and V_{Mth} are transconductances and threshold voltages. V_g is calculated from Equation (1).

Compare to physics-based FeFET models proposed in previous works (Aziz et al., 2016; Lenarczyk and Luisier, 2016), our model is more concise and friendly to the system-level simulation of SNN. Despite the simplicity, we still need to capture the timing of spikes accurately. We verify the model by utilizing it to recreate the dynamic behaviors and data provided in Wang et al. (2017). In this case, we adopt the same configuration and parameters in Wang et al. (2017), in which the FeFET is a 14 nm FinFET node that connects to a 10 nm HfO$_2$ FE layer with mode detail description in Khandelwal et al. (2017). The NMOS transistor is a FinFET but without the FE layer. For the circuits simulation, we use the default settings of $V_{DD} = 400$ mV, $V_{GM} =$ 350 mV and $C = 8$ nF. Here we use g_F= $g_M = 10^{-4}$S, $V_{Mth} = 250$ mV, and $V_g - V_{Gth} \approx 400mV$.

We simulate the circuits with varying values of V_{GF} and V_{GM} and demonstrate the results in **Figure 2**. **Figure 2A** plots two waveforms of V_S when $V_{GF} = 300$ mV and $V_{GF} = 400$ mV. It is worth noting that when $V_{GF} = 300$ mV, the hysteresis of FeFET produces normal oscillation; when $V_{GF} = 400$ mV, V_S operates between a higher range of V_{t1} and V_{t2}, which leads to a balance between the charging and discharging of capacitors and cease the oscillation. **Figure 2B** draws the I_D – V_S curves of each case, showing the FeFET's hysteretic behavior under $V_{GF} = 300$ mV. To explain the condition of oscillation, **Figure 2D** plots the flow diagram of the FeFET based oscillator. When $V_{GF} = 300$ mV, the x-axis $dV_S/dt = 0$ intersects the steep transition of the hysteretic loop. As a result, there is no attractor or fixed point but a limit cycle in the system to generate oscillations. On the other hand, when $V_{GF} = 400$ mV, the first derivative of V_S passes the charging phase of the hysteretic loop and forms a fixed point near V_S = 300 mV. The fixed point creates a stable state that eliminates the oscillation. Let us assume V_S as the membrane voltage of a neuron, its non-oscillatory state can be viewed as the resting state. The FeFET based oscillator exhibits similar dynamics as a LIF neuron, except that it fires spikes with an opposite direction. Namely, the FeFET spiking neuron fires when V_S reaches the low threshold voltage, V_{t2}, and the action potential of spikes is reversely integrated from V_{DD} to 0. Such a dynamical behavior is validated experimentally in Wang et al. (2018) (**Figures S3, S4**). If we fix V_{GF}, V_{GM} can be used to tuning the firing rate of the FeFET spiking neuron. The V_{GM} and frequency curve showed in **Figure 2C** here is measured as the instantaneous firing rate of spikes, instead of the mean frequency obtained from the power spectrum.

In summary, high V_{GF} suppress the spiking activities of the FeFET neuron and keep it at the resting state, thus exhibiting a prototypical "inhibitory" behavior. When the inhibition of V_{GF} is disabled, raising V_{GM} increases the firing rate, and the corresponding input behaves as an "excitatory" interface.

Biomimetic Neuronal Dynamics

The traditional Leaky Integrate-and-Fire (LIF) Neuron model is not able to cover the dynamics of multiple ion channels of biological neurons due to its simplicity of one dimension. Izhikevich (2003) proposed a 2-D neuron model that efficiently reproduces various dynamics of cortical neurons. The innovation of Izhikevich's model is to use a slow variable to control the leak current of a LIF model. Inspired from such a design, we propose to take advantage of inhibitory input V_{GF} in FeFET spiking neuron to imitate the function of the "slow variable" because the FeFET is responsible for the "resetting" phase (discharging) of a spike (Fang et al., 2019). Associated with the frequency adaption enabled by excitatory input V_{GM}, our neuron model can imitate multiple types of firing patterns (Fang et al., 2019). We demonstrate two types of spiking dynamics that we utilize for SNN based computation for this work. These two types of firing patterns are respectively:

- FS and LTS (Fast Spiking and Low-Threshold Spiking): firing patterns found in inhibitory cortical cells. They both feature with spike trains in high frequency. LTS has a frequency adaptation. We treat them as one

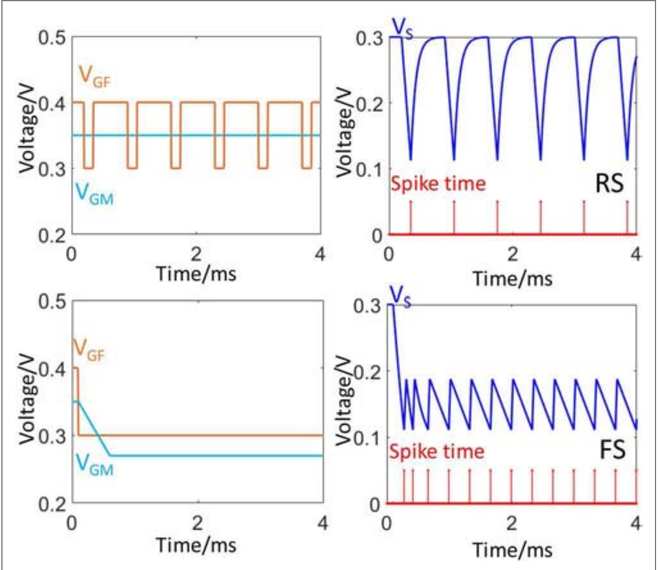


FIGURE 3 | Two neural firing patterns, FS and RS. The plots on the left column show the waveforms of input signals to V_{GF} and V_{GM}. The right column plot the waveforms of V_S (blue) and corresponding timing of spike train (red).

firing pattern (FS) for the simplicity of representation in proposed computing paradigms.

- RS (Regular Spiking): a regular cortical firing pattern with relatively low-frequency.

Figure 3 illustrates how the application of different configuration of V_{GF} and V_{GM} can generate these two firing patterns. Besides FS and RS, the FeFET spike neuron model is also capable of imitated other firing patterns such as Intrinsically Busting (IB), Chattering (CH), and interested readers are pointed to Fang et al. (2019) for further discussions. In the FS mode, the FeFET neuron operates in an oscillatory mode with disabled inhibition (low V_{GF}) for a high frequency of firing. Meanwhile, V_{GM} can be used to adjust the firing frequency. In RS mode, spikes are generated through a periodic inhibitory input which has a large duty cycle. In the original design of FeFET spiking neuron (Wang et al., 2018), the polarity of the spike train is inverted using an output inverter and the input gate voltages, V_{GF} and V_{GM} accept voltage spikes from pre-synaptic neurons via RC integrators. The two spiking modes, FS, and RS can be set by using proper input of spiking trains. **Figure S2** illustrates the frequency modulation via spikes.

Swarm Intelligence (SI)—Spiking Neural Network (SNN) Optimization

Having established the electronic equivalent of the biological neuron, we now focus on the algorithm development which can harness the dynamics of this neural circuit. In this section, we introduce the SI-SNNs that imitates the collective behavior of SI algorithms. First, we provide a general framework of SI algorithms. Then, we describe the architectures of two SI-SNNs, which are aimed at two different optimization problems, respectively.

SI Algorithm Framework

To define the problem, we use the general form of optimization, which is to find a solution of x to maximize/minimize the objective/cost function $f(x)$ under certain constraints. Namely, $x = \arg\min f(x)$, s.t constraint. For the parameter optimization of continuous objective functions, we do not take constraints into consideration.

Different SI algorithms are distinct from each other due to the different swarm behaviors they mimic. However, a general framework can be developed to fit most of these algorithmic principles. In the beginning, a swarm is initialized with multiple "agents." Each agent's location coordinates in the solution space represent the parameters of the solution. In each iteration of the optimization process, the agents move and search for solutions by updating their parameters. Such a collaboration operation is meta-heuristic and trades off between the randomization and the performance of the local search. To locate the optimal solution and to escape from local minima simultaneously, each agent follows particular behavioral rules and seek to balance exploration and exploitation (Crepinsek et al., 2011). Exploration determines the swarm's capability of discovering new candidates of the global solution. On the contrary, exploitation focuses on the individual local search within the vicinities of the current best solution. The pseudo-code in Algorithm 1 describes the framework of most SI algorithms (Fang and Dickerson, 2017).

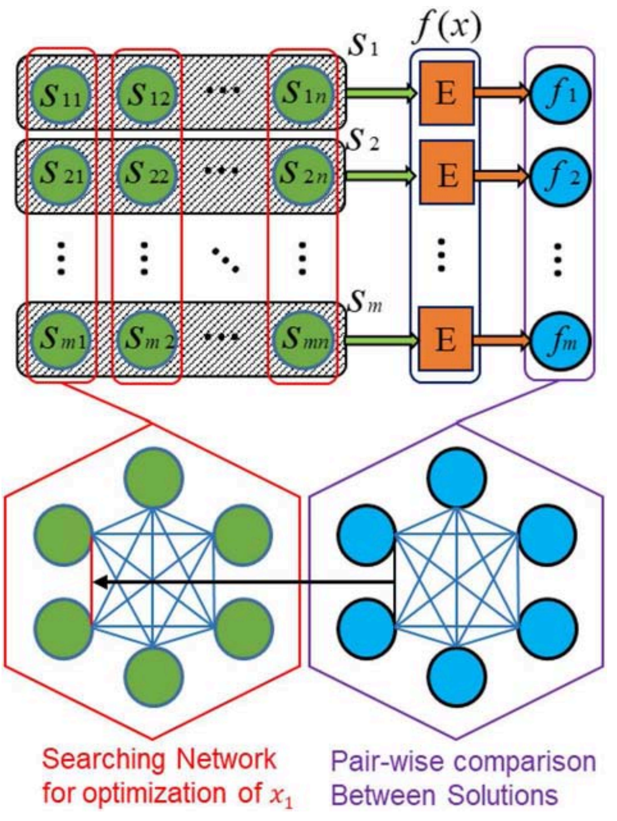
Algorithm 1: General SI Frameworks

```
1: Initialize swarm $S$ with $m$ agents $\{s1, s2, ..., sm\}$
2: While $t < MAX\_ITER$ or condition satisfy do
3:     Update vector of parameters $s_i^t = s_i^{t-1} + \Delta s_i^{t-1}$ for each
       $s_i^{t-1} \in S^{t-1}$
4:     Evaluate $f(s_i^t)$ for each $s_i^t$
5:     Compute each $\Delta s_i^t$ for the next iteration based on $f(s_i^t)$
6:     $t = t + 1$
7: end while
```

Each agent *si* in swarm *S* is an n-dimension vector that represents the variable of $f(x) \in \mathbb{R}^n \rightarrow \mathbb{R}$. The behavior rule of agent that compute Δs_i^t vary among different SI algorithms. For example, PSO updates *si* based on the history of both the best global and local solutions. FA only requires the current global best solution. Despite this distinction, SI algorithms are flexible and model-free because of their similar characteristics in meta-heuristic search. In other words, the same method can be used to address different types of optimization problems.

SI-SNN Model Architecture for Continuous Objective Function

Figure 4 depicts the architecture of the proposed SI-SNN for optimizing the parameters of continuous objective functions. Following the configuration and notation as **Algorithm 1**, we consider a swarm of m agents for an n-dimension problem. Accordingly, we prepare an $m \times n$ array of neurons (labeled as



s_1
$f(x)$
s_{11} s_{12} ⋯ s_{1n}
s_2
s_{21} s_{22} ⋯ s_{2n}
s_m
s_{m1} s_{m2} ⋯ s_{mn}
E
f_1
f_2
f_m
Searching Network for optimization of x_1
Pair-wise comparison Between Solutions

FIGURE 4 | SI-SNN architecture for parameter optimization of continuous functions.

green) to represent a parameter s_{ij} $(1 < i < m, 1 < j < n)$ in each agent s_i. The black frame with shadow encloses the neurons that belong to the **agent** s_i. The red frame indicates the neurons that compose the **searching network** for the optimization of one parameter x_k $(1 < k < n)$. Namely, each **column** of neurons is a fully connected spiking neural network defined as a searching network. Each **row** of neurons represents an agent. The block **E** (labeled as orange) evaluates the solution found by each agent by computing the value of the objective function $f(x)$. The computing platform of block E depends on the different optimization tasks and objective functions. For compatibility, it can be another spiking neural network (Iannella and Back, 2001), or a digital/mixed-signal computing hardware, or feedback from the external environment gathered through sensors such in reinforcement learning problems. The evaluation of each solution found by an individual agent produces an m-sized column vector (labeled as blue). These solutions are compared to each other and used to guide the synaptic update of the neurons.

In section Ferroelectric Based Spiking Neuron, we introduced the FeFET spiking neuron and several of its biomimetic patterns. In this scenario, we explore the use of frequency (firing rate) of each neuron to represent the value of a parameter. Therefore, an adaptable voltage-controlled high-frequency spiking mode is necessary. We choose the **FS** mode of FeFET spiking neuron (**Figure 3**), in which the inhibitory input is off ($V_{GF} = 300\,\text{mV}$) and the voltage of the capacitor V_S oscillates between $V_{t1} = 111\,\text{mV}$ and $V_{t2} = 188\,\text{mV}$. The firing rate is tuned by the excitatory input, V_{GM} (**Figure 2C**).

In a searching network, each neuron belongs to a different agent. Its firing rate represents the value of the specific parameter in the current solution. The firing rates are initialized by setting V_{GM} with random values normally distributed in a specific range. During the optimization process, these neurons adjust each other's firing rates based on the results of the pairwise comparison between solutions, following the rule described in Equation (4). For the ith neuron in a searching network, we have

$$V_{GMi} = V_{GMi} + \Delta v_{ij} + \theta\eta, \text{ on spike from j}^{\text{th}} \text{ neuron} \tag{4}$$

$$\Delta vij = \begin{cases} w(V_{GMj} - V_{GMi}), \mathit{if}\ f(s_i) < f(s_j) \\ 0, \mathit{otherwise} \end{cases}$$

where η is a Gaussian noise term and θ is a scaling factor of the stochastic term. Equation (4) explains an event-based rule of updating V_{GM}. Once a spike from the pre-synaptic neuron j arrives at the post-synaptic neuron i and if the jth agent has a better solution than the ith agent, V_{GMi} is updated by adding the difference between V_{GMi} and V_{GMj} so that it becomes more close to V_{GMj}, which reduces the difference between the firing rates of the two neurons. w is the synaptic weight that controls the step size of the V_{GM} modulation. This synaptic rule is applied to all the neurons and enables the agents with better solutions to dominate other agents by tuning their firing rate. But the dominant agents change behavior as the searching process continues. Sometimes passive agents may find better solutions as a result of a stochastic search and become active and start to modulate the neurons of other agents. The searching process ends when the neurons in every searching networks are synchronized with near-identical frequencies. Such a swarm behavior is inspired by fireflies, which attract each other via the frequency synchronization of their flash signaling (Fister et al., 2013).

SI-SNN for Traveling Salesman Problem

TSP is an NP-hard combinatorial optimization problem. Given the distance between nodes in a graph, the goal of TSP is to find a path that visits all the nodes in the graph exactly once with minimal total distance. Among SI algorithm family, ant colony optimization algorithm (ACO) was proposed to solve TSP (Dorigo and Di Caro, 1999). ACO is a swarm-based method inspired by the collaborative behavior of ants. Different from the rest of the SI algorithms, the agents (ants) in ACO do not send information to each other directly but leave the shared information (pheromone) on the edge of graphs (Dorigo and Di Caro, 1999). Individual ant makes decisions based on the concentration of pheromone on their travel route. We define a trip as complete when an agent finishes visiting all the nodes. In a trip, the amount of pheromone on the edge is updated by all the ants that have passed by that edge and further influence their choice of route in the next trip. An iteration is defined as an event when all the agents have finished one trip. After a certain

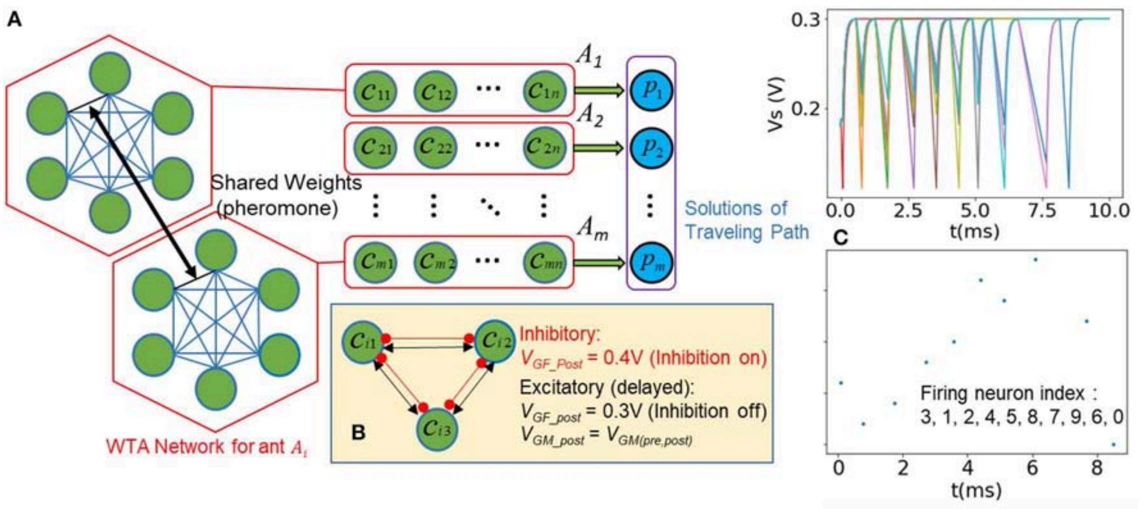

A
A_1
A_2
A_m
C_{11} C_{12} ⋯ C_{1n}
C_{21} C_{22} ⋯ C_{2n}
C_{m1} C_{m2} ⋯ C_{mn}
P_1 P_2 P_m
Shared Weights (pheromone)
Solutions of Traveling Path
WTA Network for ant A_i
B
C_{i1} C_{i2} C_{i3}
Inhibitory: V_{GF_Post} = 0.4V (Inhibition on)
Excitatory (delayed): V_{GF_post} = 0.3V (Inhibition off)
$V_{GM\_post} = V_{GM(pre,post)}$
Vs (V)
t(ms)
C
Firing neuron index : 3, 1, 2, 4, 5, 8, 7, 9, 6, 0

FIGURE 5 | (A) Modified SI-SNN architecture for solving TSP **(B)** inhibitory and excitatory synapses enable that only one neuron fires after latest pre-synaptic spike in a WTA network **(C)** Plot of state variable Vs and the raster plot that shows the order of fired spikes in one trip.

number of iterations, the best route eventually converges to the optimal solution.

Before we design the SI-SNN for ACO, we notice that a fully connected SNN with n neurons can be mapped onto a graph of n-city TSP (Hopfield and Tank, 1985) and the travel route can be indicated by the order of spikes (Jonke et al., 2016). However, the behavior of a swarm of ants is difficult to be represented simultaneously by the spike train within a single SNN. Therefore, we use multiple SNNs to simulate the trip of each ant. For each SNN, the difficulty in the design of dynamics lies on how to make each neuron fire only once and follow the correct order in one trip. In previous work (Jonke et al., 2016), multiple WTA SNNs are used to show the travel path of one trip. By exerting the inhibitory and excitatory interfaces of FeFET spiking neurons, we can use the spike train of a single SNN to represent the travel path of one agent.

Figure 5A shows the modified architecture of SI-SNN for solving TSP. We start with an $m \times n$ array of neurons (green) and each neuron represents a city (node) c_{ij} $(1 < i < m, 1 < j < n)$ in the travel path of the agent (ant) A_i. A red frame indicates a fully-connected WTA network, which models the traveling behavior of an ant A_i. In one trip, each neuron in a WTA network only fires once and the solution of the TSP p_i (labeled as blue) is represented as the order of firing of a spike train. The collaboration between agents does not rely on the evaluation of p_i. Hence, the SI-SNN architecture for ACO has no feedback loop and search networks as shown in the previous section. Instead, these WTA networks simultaneously access and update a set of shared weights that mimic the pheromone trails of the ant colony. Meanwhile, to enable the winner-takes-all mechanism, we employ an instant inhibitory synapse and a delayed excitatory synapse to pair-wise connect every neuron in the WTA network. Accordingly, we use the regular spiking (RS) mode of FeFET neuron. Namely, after the inhibition input V_{GF} was set to low, the capacitor of FeFET neuron needs to be discharged from the resting state 300 mV to the threshold voltage 111 mV to generate a spike. We describe the dynamical behavior of one WTA network (**Figure 5B**) as follow:

Step 1. The weight of pheromone τ_{ij} between any neuron i and j is initialized as 1. The inhibition of neuron is disabled ($V_{GF} =$ 300 mV). A randomly selected neuron is set as the start node with $V_{GM} = 350$ mV and the rest neurons are initialized with $V_{GM} < 350$ mV.

Step 2. The neuron of the starting node generates the first spike before the rest of the neurons reach the firing threshold and immediately set their inhibition to a high state through the inhibitory synapse, defined as ($V_{GF\_post} = 400$ mV on a pre-synaptic spike). In such a circumstance, all the neurons instantly switch to the charging stage. After they reach the resting state at 300 mV, the fired neuron will be set as inhibited till the end of the current trip, while the rest of the neurons are triggered by the delayed excitatory synapse, which is defined as:

$$\begin{cases} V_{GF\_post} = 300mV \\ V_{GM\_post} = \kappa \frac{\tau_{ij}{}^{p}}{D_{ij}{}^{q}} + \theta\eta + V_{Mth} \quad \text{(after delay } \Delta t \\ \qquad\qquad \text{on pre-synaptic spike)} \end{cases} \tag{5}$$

where the i and j are indices of pre-synaptic and postsynaptic neurons, D_{ij} is the distance between two nodes. p and q are the weights of the pheromone and the distance between the nodes, used for balancing the global and local information. κ and θ are scaling factors and η is the Gaussian random term. The rest of the neurons, which have not fired any spike yet, are free from inhibition and start to discharge (integration stage). However, their discharge rate is controlled by the

V_{GM}-, depending on the amount of pheromone, τ_{ij} and D_{ij} in Equation (5).

Step 3. The neuron that discharges the fastest become the winner, fire the second spike of this trip and inhibit other neurons. The shared weight of pheromone between the two neurons that fires in a sequence is updated as:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \frac{\omega}{D_{ij}mn} \tag{6}$$

where ρ is a decay factor, which represents the vaporization of pheromone and encourages agents to explore new routes. ω is the scaling factor of the increasing amount of pheromone.

Step 4. The whole process (Step 1 ∼3) is repeated until all the neurons in the WTA network fire a spike.

To demonstrate this process clearly, we plot the trace of V_S of neurons and the raster plot of a WTA network in **Figure 5C**. The raster plot indicates the firing order of spikes in a trip of a 10-city TSP (solution provided in **Figure 8**).

During the optimization, the process described above is executed by m WTA networks simultaneously and the pheromone trails are shared and updated on the fly. Once all the WTA networks (agents) complete a trip, a new iteration starts with the updated pheromone weights. The whole optimization process terminates when the maximum iteration number is reached.

RESULTS

Parameter Optimization of Continuous Functions

We simulate the SI-SNN computing paradigm with BRIAN, an open source SNN simulator based on Python (Stimberg et al., 2014). We use the dynamical model discussed in Section 2.2 to simulate FeFET based spiking neurons. For the first demonstration, the continuous objective function we aim at is the 2-D Schwefel's function:

$$f(x) = \sum_{i=1}^{n} \sin(\sqrt{|x_i|}) \tag{7}$$

The dimension of this function is $n = 2$, and $x_i \in [-500, 500]$. This function has more than 50 local minima and a global minimum at $\boldsymbol{x} = (418.92, 418.92)$. **Figure 6A** plots the landscape of 2-D Schwefel's function as a 3-D surface. In this case, we

TABLE 1 | Parameter optimization of benchmark objective functions.

Benchmark function (dimension)	Convergence time (Mean ± Std)	Success rate
Michalewicz's ($n = 16$)	348 ± 98 ms	89%
Schwefel's ($n = 64$)	782 ± 223 ms	92%
Ackley's ($n = 128$)	1,379 ± 928 ms	99%
De Jong's ($n = 256$)	945 ± 105 ms	100%

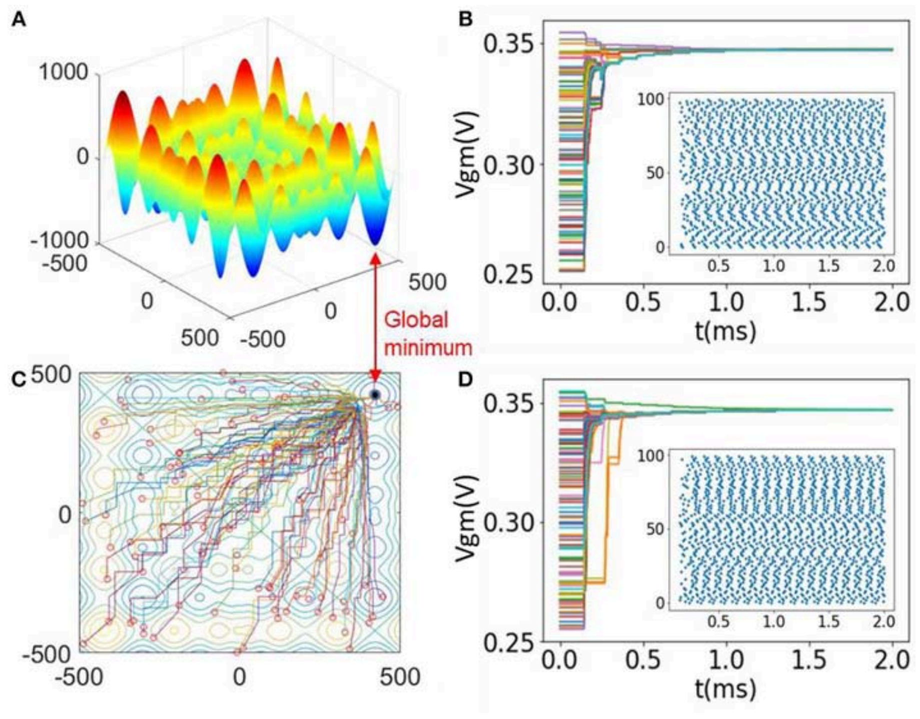


FIGURE 6 | (A) Landscape of 2-D Schwefel's function; **(B)** Contour map with solution traces of each agent in the optimization process of **(A)**; **(C,D)** Evolution of V_{GM} in two searching networks with raster plots.

prepare an SI-SNN with 100 agents and two searching networks ($m = 100$, $n = 2$). The scaling factor of random noise $\theta = 0.02$. For such a configuration, we randomly initialize the V_{GM} of each FeFET spiking neuron in the range of [255 mV, 355 mV] with a uniform distribution. Consequently, the firing rates of neurons range from 0.801 to 9.852 kHz in **FS** mode and are mapped to the range of $x_i \in [-500, 500]$. We note that when the network synchronizes, the V_{GM} of most of the neurons cluster around 339 mV and the firing rates are stabilized at 9.186 kHz. Such a value of V_{GM} corresponds to the global minima where $x_i = 418.92$. There exist errors between the parameter represented by the firing rate due to the nonlinearity in the V_{GM} - Frequency curve. It needs to be calibrated and compensated in the hardware design. In this simulation, we did not consider a hardware implementation of the evaluation blocks. **Figures 2C,D** plots the V_{GM} of each neuron in two searching networks along the optimization process. The convergence of the SI-SNN takes 1.5 ms, which is ~14 cycles of spiking. Meanwhile, we notice that the firing rates of a few of the neurons are initially attracted to local minima and then get pulled out by the neurons of other agents with better solutions. This phenomenon indicates that SI-SNN model is capable of escaping from the "trap" of local minima. **Figures 6C,D** also show the raster plots of all the spikes during the simulation process. **Figure 6B** is a contour map of **Figure 6A** with the traces of the best solutions found by each agent during the optimization. The red circles mark the initial positions of 100 agents in the solution space. Eventually the swarm converges into the global minimum.

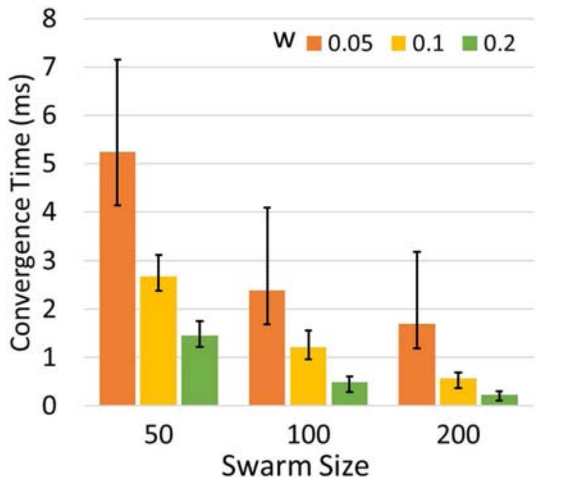


FIGURE 7 | Average convergence time to optimize 2-D Schwefel's function in different m and w. The error bars indicate the maximum and minimum time cost.

We set synaptic weight w and swarm size m to different values and run the simulation 200 times for each configuration. **Figure 7** shows the average time for the optimization problem under different configurations of w and m. The result indicates that larger m and w can speed up the optimization process. However, the best choice of w falls within a certain range. An extremely large or small value may lead to failure in synchronization or the network may miss of global optimum. Having more agents improves the efficiency and performance of optimization but also increases the demands for computing resources.

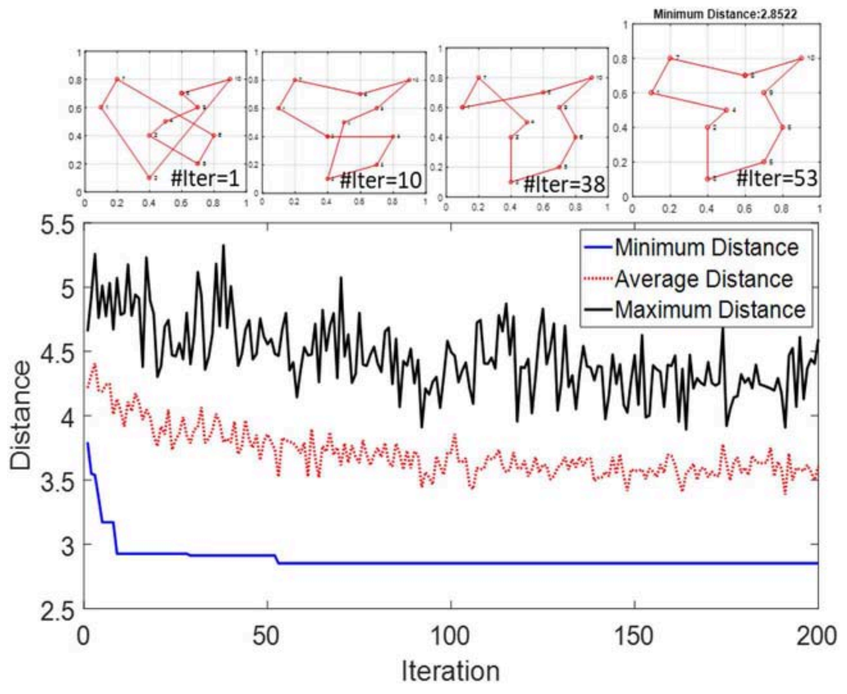


FIGURE 8 | Distance of the best solution to a 10-city TSP in each iteration of SI-SNN.

TABLE 2 | Performance of solving TSP.

TSP benchmark problem	Iteration number (Mean ± Std) (SI-SNN/Multi-SNN random Walk)	Performance (RPD %)
my10 ($n = 10$)	59 ± 28/343	0
ulysses16 ($n = 16$)	147 ± 75/1,751	0
bays29 ($n = 29$)	364 ± 143/Fail	0
att48 ($n = 48$)	625 ± 237/Fail	3%
berlin52 ($n = 52$)	574 ± 126/Fail	1%

Apart from Schwefel's function, we also test the SI-SNN on several other benchmark objective functions with different dimensions. The equations and landscape of these benchmark functions can be found in Pohlheim (2005). For the evaluation of the optimization performance, we use Relative Percentage Deviation (RPD), which we defined as the absolute percentage error between the objective function evaluation of best solution founded by algorithms and the correct optimal solution.

$$RPD = \frac{abs(f(best) - f(opt))}{f(opt)} \times 100\% \tag{8}$$

Table 1 show the average convergence time with corresponding standard deviation and the success rate in finding the near optima with an RPD smaller than 2%. In such a test, we employ swarms with 200-agent to optimize the parameter of four benchmark functions. In these simulations, we keep the same configuration of the FeFET neuron model. The time constants are the same as previous tests and the firing frequencies of neurons still range from 0.801 to 9.852 kHz. The parameters such as time and voltage, are scalable with different devices and capacitors in the FeFET based circuits, e.g., smaller capacitors may reduce the time of charge and discharge from microsecond to nanosecond (Wang et al., 2018).

Solving TSP

We use the same method to simulate the modified SI-SNN model for solving TSP. However, since the simulator does not support conditionally terminating the simulation process, we run each iteration separately in sequence. After all the WTA networks finish the trip of their agents, we reset the system and continue to run the next iteration with the updated pheromone weights. Each iteration contains $m \times n$ spikes but the time cost only depends on how fast the slowest agent fires n spikes. The whole simulation process ends when the maximum iteration number is reached. The performance and convergence speed of the original ACO are sensitive to the hyperparameters. In the simulations of this section, we set the swarm size twice as the size of the problem ($m = 2n$), $\kappa = 0.01$, $\theta = 0.03$, $\rho = 0.03$, $\omega = 2$. For q and p, it is recommended to use values within 2 and 4. However, to reduce the complexity of the hardware design, we can set both of them to 1. **Figure 8** demonstrates the optimization process of solving a 10-city TSP. It demonstrates the distances of solutions searched in each iteration and display the best route in several iterations. The optimal travel route was found at the 53rd iteration.

Next, we run a set of benchmark tests with our customized 10-city TSP and four other TSP from a standard TSP library TSPLIB. The sizes of these problems are respectively [10,16,29, 48, 52]. For each problem, we run the optimization 200 times using SI-SNN and also using SNNs that performs random-walk-based searches without any shared information (pheromone). **Table 2** shows the mean and standard deviation of iteration numbers to reach the best solution and the corresponding RPD. The standard deviation is not shown for multi-SNN random search because the successful runs are fewer than five times and such a strategy fail to find any near-optimal solution when the problem size increases. The results in **Table 2** demonstrate that without collaboration, the random search performed by a swarm is much less effective. We also notice that for complex TSPs, the SI-SNN can only approach near-optimal solutions due to the limitations inherited from the original ACO algorithm.

In **Table 3**, we estimate the "time taken" and "energy consumption" of several methods that implement ACO to solve a 48-city TSP. Bali et al. (2016) provides the performance of ACO executed respectively by a GPU and a CPU on laptop, although the 48-city TSP they use may not be att48. We conservatively estimate the energy cost of GPU and CPU based on their idle power consumption, and subtract the power consumed by the onboard memory. For the SI-SNN, we compared the time and energy cost between FeFET spiking neuron and a few of the previous literature on silicon-based neurons. We calculate the estimation results with the total spike numbers, timing, and energy cost per spike. In this scenario, we do not consider the delay and power consumption of synapses and assume the neurons of previous works is also compatible with the WTA network in SI-SNN. For FeFET based spiking neurons, we provide two sets of data, 45 nm FinFET process with $C = 8$ nF and 14 nm FinFET process with $C = 1$ pF. The first one has a relatively lower frequency in the kHz range and higher energy consumption of ~0.36 nJ/spike. The second one uses a predictive transistor technology and a smaller capacitor that generates oscillation frequency in the MHz range. The comparison in **Table 3** shows that the FeFET based SI-SNN is a promising computing paradigm for optimization in terms of high performance and energy efficiency. Even with traditional CMOS, event-based SI-SNN is highly energy efficient compared to CMOS digital systems. Compared with silicon neurons, we observe that post-CMOS, emerging devices can effectively reduce the number of transistors as well by harnessing the inherent neuronal dynamics. In particular, the FeFET spiking neuron provides both excitatory and inhibitory interfaces, which benefits the design of the WTA network. It reduces the number of neurons and synapses. For example, without inhibition input directly to the neuron, representing one trip of N-city TSP requires $N \times N$ neuron (Jonke et al., 2016), while we only use a single N-neuron WTA network in this work. Thus, the energy reduction brought by the unique feature of FeFET spiking neuron is not shown in **Table 3**.

TABLE 3 | Comparison of proposed computing paradigm and other methods.

Task: ACO for 48-city TSP	**SI-SNN**				**GPU**	**CPU**
References	**Indiveri (2003)**	**Wijekoon and Dudek (2008)**	**Babacan et al. (2016)**	**Wang et al. (2017, 2018) used in this work**	**Bali et al. (2016)**	**Bali et al. (2016)**
Technology	CMOS analog	CMOS analog	Memristor+CMOS	FeFET+CMOS	CMOS digital	CMOS digital
Manufacturing process	1.5 μm	0.35 μm	0.18 μm	45 nm (14 nm)	40 nm	22 nm
Neuron model (Dynamics)	LIF (RS)	Izhkevich (RS, FS, CH, IB)	Izhkevich (RS, FS, CH, IB)	Izhkevich (RS, FS, CH, IB)	GPU model: GTX 480M	CPU model: Intel Core T i7-4700MQ
Device count/neuron (total)	18T (172.8k)	14T (134.4k)	1T+3M (38.4k)	1T+1FeFET (19.2k)	3 billion on chip	1.4 billion on chip
Synaptic input	Excitatory	Excitatory	Excitatory	**Excitatory+Inhibitory**	/	/
Time cost	5 min	36 ms	3 s	3.9 s (0.48 ms)	2.4 s	6.8 s
Energy consumption of neuron in total	90 mJ	0.1 mJ	0.5 mJ	2.2 mJ (∼nJ)	∼190J	∼320J

DISCUSSION

In this paper we propose SI-SNN as a computational platform based on FeFET based spiking neurons. We observe that:

1. The FeFET based spiking neurons exhibit rich neuronal dynamics. In the SI-SNN architecture, we use the rate-based representation in the FS mode for the optimization of the continuous objective function and the phase-based representation in the RS mode for solving TSP. To the best of our knowledge this is one of the first demonstrations of a computing platform that harnesses various neuronal dynamics for solving different optimization problems.
2. The inhibitory input of FeFET spiking neuron facilitates the design of the WTA network in solving TSP. In our design, the spiking behavior of neurons can inhibit and compete with each other, and naturally mimic path planning of ants. Without the inhibitory interface, more hardware resources are required.
3. The design of FeFET spiking neuron is compact. The entire circuit can run at high frequency with low energy cost.
4. The dynamical behavior model we extract is simple and effective. It can capture the spike timing but bypass the complex physical equations of ferroelectric devices, and improve the efficiency of the simulation.

Given the simulation results of the first SI-SNN model in section Parameter Optimization of Continuous Functions, we observe two tradeoffs between the metrics of continuous function optimizations. The first one is between the spatial cost and the temporal cost. A larger size of a swarm results in faster speed of convergence but also requires more neurons and spike generators, which is equivalent to the tradeoff between efficiency and energy. The second one is between convergence speed and accuracy. A larger network weight and less randomization may improve the efficiency of the search process but also increases the risk of missing the optima. In particular, the random term in metaheuristic search becomes increasingly important as the problem dimension increases, because the search routine covers less of a solution space in a higher dimension. These observations can be used to tune model parameters.

In the SI-SNN TSP solver, our design benefits from the dynamical feature of FeFET based spiking neurons. The excitatory and inhibitory interfaces enable the design of the WTA embedded in each SNN. The simulation results emphasize the importance of shared information between agents in the collaborative search process of swarms. Further work can be pursued by invoking more ACO algorithms such as Max-min ant systems (MMAS) (Stützle and Hoos, 2000) and ant colony system (ACS) (Dorigo and Gambardella, 1997) that can improve the performance and convergence speed at the cost of more complicated hardware design.

As far as the hardware implementation is concerned, the solution-based adaption of synaptic parameters can be realized with address-event representation (AER) systems (Park et al., 2012) or memristor crossbar arrays (Long et al., 2016; Ielmini, 2018). The random terms in the synaptic rule can be implemented via the emerging stochastic devices such as spintronic device and memristors (Vincent et al., 2015). Furthermore, future works may harness more learning properties from synapse models with non-linear dynamics. Also, the interplay between swarm intelligence and individual cognitive intelligence is a research area that remains active (Rosenberg et al., 2016). The results will have contributions to fields as varied as multi-agent artificial intelligence, social psychology, cognitive science and so on.

In summary, we propose a new SNN computing paradigm built on FeFET spiking neuron that combines swarm intelligence in agents of spiking neural network to address optimization problems. We simulate our SI-SNN model with SNN simulator and demonstrate its capability to optimizing parameters of continuous objective functions and for solving the traveling salesman problem. In our design, we utilize two types of neural dynamics, FS and RS, to encode information with firing rate and spike timing, respectively, to perform varying computational tasks. The FeFET based SNN is a promising hardware platform for achieving the energy-efficiency and high-performance denoted by future computing systems (Wang et al., 2018). We demonstrate the computational power of neuromorphic systems in the field of general optimization problems. Above all, our work sheds light on the connection between individual intelligence and swarm intelligence.

DATA AVAILABILITY

No datasets were generated or analyzed for this study.

AUTHOR CONTRIBUTIONS

YF proposed the method of SI-SNN and performed the simulation and data analysis. AR and YF formulate the problem and drafted the manuscript. JG, ZW, SD, and AK worked on the device and circuits of FeFET spiking neuron.

FUNDING

This work was supported by ASCENT and C-BRIC, two of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2019.00855/full#supplementary-material

REFERENCES

Aziz, A., Breyer, E. T., Chen, A., Chen, X., Datta, S., Gupta, S. K., et al. (2018). "Computing with ferroelectric FETs: devices, models, systems, and applications," in *Proceedings of IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)* (Washington, DC: IEEE).

Aziz, A., Ghosh, S., Datta, S., and Gupta, S. K. (2016). Physics-based circuit-compatible SPICE model for ferroelectric transistors. *IEEE Electron Device Lett.* 37, 805–808. doi: 10.1109/LED.2016.2558149

Babacan, Y., Kaçar, F., and Gürkan, K. (2016). A spiking and bursting neuron circuit based on memristor. *Neurocomputing* 203, 86–91. doi: 10.1016/j.neucom.2016.03.060

Bali, O., Elloumi, W., Abraham, A., and Alimi, A. M. (2016). "ACO-PSO optimization for solving TSP problem with GPU acceleration," in *International Conference on Intelligent Systems Design and Applications* (Cham: Springer).

Blum, C., and Li, X. (2008). "Swarm intelligence in optimization," in *Swarm Intelligence*, eds C. Blum and D. Merkle (Berlin, Heidelberg: Springer), 43–85.

Bouganis, A., and Shanahan, M. (2010). "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity," in *The 2010 International Joint Conference on Neural Networks (IJCNN)* (Washington, DC: IEEE).

Brody, C. D., and Hopfield, J. J. (2003). Simple networks for spike-timing-based computation, with application to olfactory processing. *Neuron* 37, 843–852. doi: 10.1016/S0896-6273(03)00120-X

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Computer Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Crepinsek, M., Mernik, M., and Liu, S. H. (2011). Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees. *Int. J. Innovat. Comput. Appl.* 3, 11–19. doi: 10.1504/IJICA.2011.037947

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., and Liao, Y., et al (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. (2016). "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *Proceedings of IEEE International Conference Rebooting Computing (ICRC)* (Washington, DC: IEEE).

Dorigo, M., and Di Caro, G. (1999). "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)* (Washington, DC: IEEE).

Dorigo, M., and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transac. Evol. Comput.* 1, 53–66. doi: 10.1109/4235.585892

Duan, H., and Luo, Q. (2015). New progresses in swarm intelligence–based computation. *Int. J. Bio-Inspired Comput.* 7, 26–35. doi: 10.1504/IJBIC.2015.067981

Fang, Y., and Dickerson, S. J. (2017). "Achieving swarm intelligence with spiking neural oscillators," in *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Washington, DC: IEEE).

Fang, Y., Gomez, J., Wang, Z., Datta, S., Khan, A. I., and Raychowdhury, A. (2019). Neuro-mimetic dynamics of a ferroelectric FET based spiking neuron. *IEEE Electron Device Lett.* 40, 1213–1216. doi: 10.1109/LED.2019.2914882

Fang, Y., Yashin, V. V., Seel, A. J., Jennings, B., Barnett, R., Chiarulli, D. M., et al. (2014). "Modeling oscillator arrays for video analytic applications," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design* (Washington, DC: IEEE).

Fister, I., Fister I. Jr., Yang, X. S., and Brest, J. (2013). A comprehensive review of firefly algorithms. *Swarm Evol. Comput.* 13, 34–46. doi: 10.1016/j.swevo.2013.06.001

Fonseca Guerra, G. A., and Furber, S. B. (2017). Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Front. Neurosci.* 11:714. doi: 10.3389/fnins.2017.00714

Ghosh-Dastidar, S., and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308. doi: 10.1142/S0129065709002002

Goss, S., Aron, S., Deneubourg, J. L., and Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* 76, 579–581. doi: 10.1007/BF00462870

Guevara Erra, R., Perez Velazquez, J. L., and Rosenblum, M. (2017). Neural synchronization from the perspective of non-linear dynamics. *Front. Computat. Neurosci.* 11:98. doi: 10.3389/fncom.2017.00098

Hopfield, J. J., and Tank, D. W. (1985). "Neural" computation of decisions in optimization problems. *Biol. Cybernet.* 52, 141–152.

Iannella, N., and Back, A. D. (2001). A spiking neural network architecture for nonlinear function approximation. *Neural Netw.* 14, 933–939. doi: 10.1016/S0893-6080(01)00080-6

Ielmini, D. (2018). Brain-inspired computing with resistive switching memory (RRAM): devices, synapses and neural networks. *Microelectronic Eng.* 190, 44–53. doi: 10.1016/j.mee.2018.01.009

Indiveri, G. (2003). "A low-power adaptive integrate-and-fire neuron circuit," in *Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, ISCAS'03* (Washington, DC: IEEE).

Indiveri, G., and Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. *Front. Neurosci.* 5:118. doi: 10.3389/fnins.2011.00118

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., Linares-Barranco, B., Legenstein, R., Deligeorgis, G., and Prodromakis, T. (2013). Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology* 24:384010. doi: 10.1088/0957-4484/24/38/384010

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transac. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440

Jonke, Z., Habenschuss, S., and Maass, W. (2016). Solving constraint satisfaction problems with networks of spiking neurons. *Front. Neurosci.* 10:118. doi: 10.3389/fnins.2016.00118

Kasabov, N. K. (2014). NeuCube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Netw.* 52, 62–76. doi: 10.1016/j.neunet.2014.01.006

Kennedy, J., and Eberhart, R. C. (1999). "The particle swarm: social adaptation in information-processing systems," in *New Ideas in Optimization*, eds D. Corne, M. Dorigo, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price (Maidenhead: McGraw-Hill Ltd.), 379–388.

Khandelwal, S., Duarte, J. P., Khan, A. I., Salahuddin, S., and Hu, C. (2017). Impact of parasitic capacitance and ferroelectric parameters on negative capacitance FinFET characteristics. *IEEE Electron Device Lett.* 38, 142–144. doi: 10.1109/LED.2016.2628349

Kuzum, D., Yu, S., and Wong, H. P. (2013). Synaptic electronics: materials, devices and applications. *Nanotechnology* 24:382001. doi: 10.1088/0957-4484/24/38/382001

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521:436. doi: 10.1038/nature14539

Lenarczyk, P., and Luisier, M. (2016). "Physical modeling of ferroelectric field-effect transistors in the negative capacitance regime," in *IEEE International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)* (Washington, DC: IEEE).

Long, Y., Jung, E. M., Kung, J., and Mukhopadhyay, S. (2016). "Reram crossbar based recurrent neural network for human activity detection," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Washington, DC: IEEE).

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Mead, C. A. (1989). *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Michener, C. D. (1969). Comparative social behavior of bees. *Ann. Rev. Entomol.* 14, 299–342. doi: 10.1146/annurev.en.14.010169.001503

Mostafa, H., Müller, L. K., and Indiveri, G. (2015). An event-based architecture for solving constraint satisfaction problems. *Nat. Commun.* 6:8941. doi: 10.1038/ncomms9941

Norris, K. S., and Schilt, C. R. (1988). Cooperative societies in three-dimensional space: on the origins of aggregations, flocks, and schools, with special reference to dolphins and fish. *Ethol. Sociobiol.* 9, 149–179. doi: 10.1016/0162-3095(88)90019-2

Parihar, A., Jerry, M., Datta, S., and Raychowdhury, A. (2018). Stochastic IMT (insulator-metal-transition) neurons: an interplay of thermal and threshold noise at bifurcation. *Front. Neurosci.* 12:210. doi: 10.3389/fnins.2018.00210

Parihar, A., Shukla, N., Jerry, M., Datta, S., and Raychowdhury, A. (2017). Vertex coloring of graphs via phase dynamics of coupled oscillatory networks. *Sci. Rep.* 7:911. doi: 10.1038/s41598-017-00825-1

Park, J., Yu, T., Maier, C., Joshi, S., and Cauwenberghs, G. (2012). "Live demonstration: hierarchical address-event routing architecture for reconfigurable large scale neuromorphic systems," in *2012 IEEE International Symposium on Circuits and Systems* (Washington, DC: IEEE).

Pohlheim, H. (2005). "Geatbx examples examples of objective functions," in *Documentation for GEATbx version 3.7 (Genetic and Evolutionary Algorithm Toolbox for use with Matlab)* (Washington, DC: IEEE).

Ponulak, F., and Kasinski, A. (2011). Introduction to spiking neural networks: information processing, learning and applications. *Acta Neurobiol. Exp.* 71, 409–433. Available online at: https://ane.pl/archive?vol=71&no=4&id=7146

Romera, M., Talatchian, P., Tsunegi, S., Araujo, F. A., Cros, V., Bortolotti, P., et al. (2018). Vowel recognition with four coupled spin-torque nano-oscillators. *Nature* 563:230. doi: 10.1038/s41586-018-0632-y

Rosenberg, L., Baltaxe, D., and Pescetelli, N. (2016). "Crowds vs swarms, a comparison of intelligence," in *2016 Swarm/Human Blended Intelligence Workshop (SHBI)* (Washington, DC: IEEE).

Stimberg, M., Goodman, D. F., Benichoux, V., and Brette, R. (2014). Equation-oriented specification of neural models for simulations. *Front. Neuroinformatics* 8:6. doi: 10.3389/fninf.2014.00006

Stützle, T., and Hoos, H. H. (2000). MAX–MIN ant system. *Future Generat. Computer Syst.* 16, 889–914. doi: 10.1016/S0167-739X(00)00043-1

Vincent, A. F., Larroque, J., Locatelli, N., Romdhane, N. B., Bichler, O., Gamrat, C., et al. (2015). Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems. *IEEE Transac. Biomed. Circuits Syst.* 9, 166–174. doi: 10.1109/TBCAS.2015.24 14423

Wang, Z., Crafton, B., Gomez, J., Xu, R., Luo, A., Krivokapic, Z., et al. (2018). "Experimental demonstration of ferroelectric spiking neurons for unsupervised clustering," in *2018 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC: IEEE).

Wang, Z., Khandelwal, S., and Khan, A. I. (2017). Ferroelectric oscillators and their coupled networks. *IEEE Electron Device Lett.* 38, 1614–1617. doi: 10.1109/LED.2017.2754138

Wijekoon, J. H., and Dudek, P. (2008). Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Netw.* 21, 524–534. doi: 10.1016/j.neunet.2007. 12.037

Yang, X. S. (2010). "A new metaheuristic bat-inspired algorithm," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)* (Berlin, Heidelberg: Springer).

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ORIGINAL RESEARCH
published: 27 August 2019
doi: 10.3389/fnins.2019.00883

Reinforcement Learning With Low-Complexity Liquid State Machines

Wachirawit Ponghiran†, Gopalakrishnan Srinivasan† and Kaushik Roy*

Department of ECE, Purdue University, West Lafayette, IN, United States

We propose reinforcement learning on simple networks consisting of random connections of spiking neurons (both recurrent and feed-forward) that can learn complex tasks with very little trainable parameters. Such sparse and randomly interconnected recurrent spiking networks exhibit highly non-linear dynamics that transform the inputs into rich high-dimensional representations based on the current and past context. The random input representations can be efficiently interpreted by an output (or readout) layer with trainable parameters. Systematic initialization of the random connections and training of the readout layer using Q-learning algorithm enable such small random spiking networks to learn optimally and achieve the same learning efficiency as humans on complex reinforcement learning (RL) tasks like Atari games. In fact, the sparse recurrent connections cause these networks to retain fading memory of past inputs, thereby enabling them to perform temporal integration across successive RL time-steps and learn with partial state inputs. The spike-based approach using small random recurrent networks provides a computationally efficient alternative to state-of-the-art deep reinforcement learning networks with several layers of trainable parameters.

Keywords: liquid state machine, recurrent SNN, learning without stable states, spiking reinforcement learning, Q-learning

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Sadique Sheik,
AiCTX AG, Switzerland
Arash Ahmadi,
University of Windsor, Canada

****Correspondence:***
Wachirawit Ponghiran
wponghir@purdue.edu

†These authors have contributed equally to this work

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *10 April 2019*
Accepted: *07 August 2019*
Published: *27 August 2019*

Citation:
Ponghiran W, Srinivasan G and Roy K (2019) Reinforcement Learning With Low-Complexity Liquid State Machines. Front. Neurosci. 13:883. doi: 10.3389/fnins.2019.00883

1. INTRODUCTION

High degree of recurrent connectivity among neuronal populations is a key attribute of neural microcircuits in the cerebral cortex and many different brain regions (Douglas et al., 1995; Harris and Mrsic-Flogel, 2013; Jiang et al., 2015). Such common structure suggests the existence of a general principle for information processing. However, the principle underlying information processing in such recurrent population of spiking neurons is still largely elusive due to the complexity of training large recurrent Spiking Neural Networks (SNNs). In this regard, reservoir computing architectures (Maass et al., 2002, 2003; Lukoševičius and Jaeger, 2009) were proposed to minimize the training complexity of large recurrent neuronal populations. Liquid State Machine (LSM) (Maass et al., 2002, 2003) is a recurrent SNN consisting of an input layer sparsely connected to a randomly interlinked reservoir (or liquid) of spiking neurons whose activations are passed on to a readout (or output) layer, trained using supervised algorithms, for inference. The key attribute of an LSM is that the input-to-liquid and the recurrent excitatory $\leftrightarrow$ inhibitory synaptic connectivity matrices and weights are fixed *a priori*. LSM effectively utilizes the rich non-linear dynamics of Leaky-Integrate-and-Fire spiking neurons (Dayan and Abbott, 2003) and the sparse random input-to-liquid and recurrent-liquid synaptic connectivity for processing spatio-temporal inputs. At any time instant, the spatio-temporal inputs are transformed into a high-dimensional representation, referred to as the liquid states (or spike patterns), which evolves dynamically based on decaying memory of the past inputs. The memory capacity of the liquid is dictated by its size and degree of recurrent connectivity. Although the LSM, by construction, does not have stable instantaneous internal states like Turing machines (Savage, 1998) or attractor neural networks (Amit, 1992), prior studies have successfully trained the readout layer using liquid activations, estimated by integrating the liquid states (spikes) over time, for speech recognition (Auer et al., 2002; Maass et al., 2002; Verstraeten et al., 2005; Bellec et al., 2018), image recognition (Srinivasan et al., 2018), gesture recognition (Chrol-Cannon and Jin, 2015; Panda and Srinivasa, 2018), and sequence generation tasks (Nicola and Clopath, 2017; Panda and Roy, 2017; Bellec et al., 2019).

In this work, we propose such sparse randomly-interlinked low-complexity LSMs for solving complex Reinforcement Learning (RL) tasks, which involve an autonomous agent (modeled using the LSM) trained to select actions in a manner that maximizes the expected future rewards received from the environment. For instance, a robot (agent) learning to navigate a maze (environment) based on the reward and punishment received from the environment is an example RL task. The environment state (converted to spike trains) is fed to the liquid, which produces a high-dimensional representation based on current and past inputs. The sparse recurrent connections enable the liquid to retain decaying memory of past input representations and perform temporal integration across different RL time-steps. We present an optimal initialization strategy for the fixed input-to-liquid and recurrent-liquid connectivity matrices and weights to enable the liquid to produce high-dimensional representations that lead to efficient training of the liquid-to-readout weights. Artificial rate-based neurons for the readout layer takes the liquid activations and produces *action-values* to guide action selection for a given environment state. The liquid-to-readout weights are trained using the Q-learning RL algorithm proposed for deep learning networks (Mnih et al., 2015). In RL theory (Sutton and Barto, 1998), the Q-value, also known as the action-value, estimates the expected future rewards for a state-action pair that specifies how good is the action for the current environment state. The readout layer of the LSM contains as many neurons as the number of possible actions for a particular RL task. At any given time, the readout neurons predict the Q-value for all possible actions based on the high-dimensional state representation provided by the liquid. The liquid-to-readout weights are then trained using backpropagation (Rumelhart et al., 1986) to minimize the error between the Q-values predicted by the LSM and the target Q-values estimated from RL theory (Watkins and Dayan, 1992) as described in subsection 2.2. We adopt ϵ-greedy policy (explained in subsection 2.2) to select the suitable action based on the predicted Q-values during training and evaluation. Based on ϵ-greedy policy, a lot of random actions are picked in the beginning of the training phase to better explore the environment. Toward the end of training and during inference, the action corresponding to the maximum Q-value is selected with higher probability to exploit the learnt experiences. We first demonstrate the utility of the sparse recurrent connections in enabling the LSM to perform temporal integration across RL time-steps by training it to perform the Cartpole-balancing RL task (Sutton and Barto, 1998) with partial state inputs. We feed only the cart position and pole angle to the LSM while suppressing the cart velocity and angular velocity of the pole. We show that the fading memory of the past cart position and pole angle retained by the liquid enables it to make better decisions without the velocity information compared to an LSM without recurrent connections. We then comprehensively validate the capability of the LSM and the presented training methodology on complex RL tasks like Pacman (DeNero et al., 2010) and Atari games (Brockman et al., 2016). We note that LSM has been previously trained using Q-learning for RL tasks pertaining to robotic motion control (Joshi and Maass, 2005; Berberich, 2017; Tieck et al., 2018). We demonstrate and benchmark the efficacy of appropriately initialized LSM for solving RL tasks commonly used to evaluate deep reinforcement learning networks. In essence, this work provides a promising step toward incorporating bio-plausible low-complexity recurrent SNNs like LSMs for complex RL tasks, which can potentially lead to much improved energy efficiency in event-driven asynchronous neuromorphic hardware implementations (Merolla et al., 2014; Davies et al., 2018).

2. MATERIALS AND METHODS

2.1. Liquid State Machine: Architecture and Initialization

Liquid State Machine (LSM) consists of an input layer sparsely connected via fixed synaptic weights to a randomly interlinked liquid of spiking neurons followed by a readout layer as depicted in **Figure 1**. Each spiking neuron fires an action potential that leads to either excitatory or inhibitory effect at all of its termination sites. Based on the terminology followed in Maass et al. (2002) and Diehl and Cook (2015), we term a neuron that leads to excitatory (inhibitory) effect an excitatory (inhibitory) neuron. The input layer (denoted by P) is modeled as a group of excitatory neurons that spike based on the input environment state following a Poisson process. The sparse input-to-liquid connections are initialized such that each excitatory neuron in the liquid receives synaptic connections from approximately K random input neurons. This guarantees uniform excitation of the liquid-excitatory neurons by the external input spikes. The fixed input-to-liquid synaptic weights are chosen from a uniform distribution between 0 and α as shown in **Table 1**, where α is the maximum bound imposed on the weights. The liquid consists of excitatory neurons (denoted by E) and inhibitory neurons (denoted by I) recurrently connected in a sparse random manner as illustrated in **Figure 1**. The number of excitatory neurons is chosen to be 4× the number of inhibitory neurons as observed in the cortical circuits (Wehr and Zador, 2003). We use the Leaky-Integrate-and-Fire (LIF) model (Dayan and Abbott, 2003) to mimic the dynamics of both excitatory and inhibitory spiking neurons as described by the following differential equations:

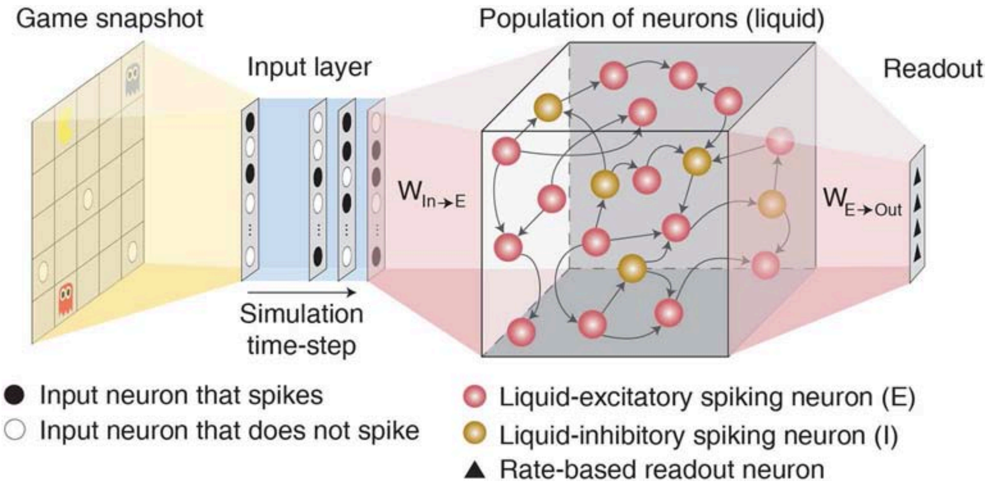

Game snapshot
Input layer
Population of neurons (liquid)
Readout
$W_{In\to E}$
$W_{E\to Out}$
Simulation time-step
● Input neuron that spikes
○ Input neuron that does not spike
Liquid-excitatory spiking neuron (E)
Liquid-inhibitory spiking neuron (I)
▲ Rate-based readout neuron

FIGURE 1 | Illustration of the LSM architecture consisting of an input layer sparsely connected via fixed synaptic weights to randomly recurrently connected reservoir (or liquid) of excitatory and inhibitory spiking neurons followed by a readout layer composed of artificial rate-based neurons.

TABLE 1 | Synaptic weight initialization parameters for the fixed LSM connections for learning to balance cartpole, play Pacman, and play Atari game.

Connection type	Weight
INPUT-TO-LIQUID CONNECTIONS	
$P \to E$	[0, 0.6]
RECURRENT-LIQUID CONNECTIONS	
$E \to E$	[0, 0.05]
$E \to I$	[0, 0.25]
$I \to E$	[0, 0.3]
$I \to I$	[0, 0.01]

TABLE 2 | Leaky-Integrate-and-Fire (LIF) model parameters for the liquid neurons.

Parameter	Value
EXCITATORY AND INHIBITORY NEURONS	
V_{rest}	0
V_{reset}	0
V_{thres}	0.5
τ	20 ms
τ_{refrac}	1 ms
Δt (simulation time-step)	1 ms

$$\frac{dV_i}{dt} = \frac{V_{rest} - V_i}{\tau} + I_i(t) \tag{1}$$

$$I_i(t) = \sum_{l \in N_P} W_{li} \cdot \delta(t-t_l) + \sum_{j \in N_E} W_{ji} \cdot \delta(t-t_j) - \sum_{k \in N_I} W_{ki} \cdot \delta(t-t_k) \tag{2}$$

where V_i is the membrane potential of the i-th neuron in the liquid, V_{rest} is the resting potential to which V_i decays to, with time constant τ, in the absence of input current, and $I_i(t)$ is the instantaneous current projecting into the i-th neuron, and N_P, N_E, and N_I are the number of input, excitatory, and inhibitory neurons, respectively. The instantaneous current is a sum of three terms: current from input neurons, current from excitatory neurons, and current from inhibitory neurons. The first term integrates the sum of pre-synaptic spikes, denoted by $\delta(t - t_l)$ where t_l is the time instant of pre-spikes, with the corresponding synaptic weights (W_{li} in Equation 2). Likewise, the second (third) term integrates the sum of pre-synaptic spikes from the excitatory (inhibitory) neurons, denoted by $\delta(t - t_j)$ ($\delta(t - t_k)$), with the respective weights W_{ji} (W_{ki}) in Equation 2. The neuronal membrane potential is updated with the sum of the input, excitatory, and negative inhibitory currents as shown in Equation 1. When the membrane potential reaches a certain threshold V_{thres}, the neuron fires an output spike. The membrane potential is thereafter reset to V_{reset} and the neuron is restrained from spiking for an ensuing refractory period by holding its membrane potential constant. The LIF model hyperparameters for the excitatory and inhibitory neurons are listed in **Table 2**.

There are four types of recurrent synaptic connections in the liquid, namely, $E \to E$, $E \to I$, $I \to E$, and $I \to I$. We express each connection in the form of a matrix that is initialized to be sparse and random, which causes the spiking dynamics of a particular neuron to be independent of most other neurons and maintains separability in the neuronal spiking activity. However, the degree of sparsity needs to be tuned to achieve rich network dynamics. We find that excessive sparsity (reduced connectivity) leads to weakened interaction between the liquid neurons and renders the liquid memoryless. On the contrary, lower sparsity (increased connectivity) results in chaotic spiking activity, which eliminates the separability in neuronal spiking activity. We initialize the connectivity matrices such that each excitatory neuron receives approximately C synaptic connections from inhibitory neurons, and vice versa. The hyperparameter C is tuned empirically as discussed in subsection 3.1 to avoid common chaotic spiking activity problems that occur when (1) excitatory neurons connect to each other and form a loop that always leads to positive drift in membrane potential, and when (2) an excitatory neuron connects to itself and repeatedly gets excited from its activity. Specifically, for the first situation, we have non-zero elements in the connectivity matrix $E \rightarrow E$ (denoted by W_{EE}) only at locations where elements in the product of connectivity matrices $E \rightarrow I$ and $I \rightarrow E$ (denoted by W_{EI} and W_{IE}, respectively) are non-zero. This ensures that excitatory synaptic connections are created only for those neurons that also receive inhibitory synaptic connections, which mitigates the possibility of continuous positive drift in the respective membrane potentials. To circumvent the second situation, we force the diagonal elements of W_{EE} to be zero and eliminate the possibility of repeated self-excitation. Throughout this work, we create a recurrent connectivity matrix for liquid with m excitatory neurons and n inhibitory neurons by forming an $m \times n$ matrix whose values are randomly drawn from a uniform distribution between 0 and 1. Connection is formed between those pairs of neurons where the corresponding matrix entries are lesser than the target connection probability ($= C/m$). For illustration, consider a liquid with $m = 1,000$ excitatory and $n = 250$ inhibitory neurons. In order to create the $E \rightarrow I$ connectivity matrix such that each inhibitory neuron receives synaptic connection from a single excitatory neuron ($C = 1$), we first form a $1,000 \times 250$ random matrix whose values are drawn from a uniform distribution between 0 and 1. We then create a connection between those pairs of neurons where the matrix entries are lesser than 0.1% (1/1,000). Similar process is repeated for connection $I \rightarrow E$. We then initialize connection $E \rightarrow E$ based on the product of W_{EI} and W_{IE}. Similarly, the connectivity matrix for $I \rightarrow I$ (denoted by W_{II}) is initialized based on the product of W_{IE} and W_{EI}. The connection weights are initialized from a uniform distribution between 0 and β as shown in **Table 1** for different recurrent connectivity matrices, unless stated otherwise. Note that the weights of the synaptic connections from inhibitory neurons are greater than that for synaptic connections from excitatory neurons to account for the lower number of inhibitory neurons relative to excitatory neurons. Stronger inhibitory connection weights help ensure that every neuron receives similar amount of excitatory and inhibitory input currents, which improves the stability of the liquid as experimentally validated in subsection 3.1.

The liquid-excitatory neurons are fully-connected to artificial rate-based neurons in the readout layer for inference. The readout layer, which consists of as many output neurons as the number of actions for a given RL task, uses the average firing rate/activation of the excitatory neurons to predict the Q-value for every state-action pair. We translate the liquid spiking activity to average rate by accumulating the excitatory neuronal spikes over the time period for which the input (current environment state) is presented. We then normalize the spike counts with the maximum possible spike count over the LSM-simulation period, which is computed as the LSM-simulation period divided by the simulation time-step, to obtain the average firing rate of the excitatory neurons that are fed to the readout layer. Since the number of excitatory neurons is larger than the number of output neurons in the readout layer, we gradually reduce the dimension by introducing an additional fully-connected hidden layer between the liquid and the output layer. We use ReLU non-linearity (Nair and Hinton, 2010) after the first hidden layer but none after the final output layer since the Q-values are unbounded and can assume positive or negative values. We train the synaptic weights constituting the fully-connected readout layer using the Q-learning based training methodology that is described in the following subsection 2.2.

2.2. Q-Learning Based LSM Training Methodology

At any time instant t in RL task, the agent receives the environment state s_t and picks action a_t from the set of all possible actions. After the environment receives the action a_t, it transitions to the next state based on the chosen action and feeds back an immediate reward r_{t+1} and the new environment state s_{t+1}. As mentioned in the beginning, the goal of the agent is to maximize the accumulated reward in the future, which is mathematically expressed as

$$R_t = \sum_{t=1}^{\infty} \gamma^t \, r_t \tag{3}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the relative significance attributed to immediate and future reward. If γ is chosen to be 0, the agent maximizes only the immediate reward. However, as γ approaches unity, the agent learns to maximize the accumulated reward in the future. Q-learning (Watkins and Dayan, 1992) is a widely used RL algorithm that enables the agent to achieve this objective by computing the state-action value function (or commonly known as the Q-function), which is the expected future reward for a state-action pair that is specified by

$$Q_\pi(s, a) = \mathrm{E}[R_t | s_t = s, a_t = a, \pi] \tag{4}$$

where $Q_\pi(s, a)$ measures the value of choosing an action a when in state s following a policy π. If the agent follows the optimal policy (denoted by π_*) such that $Q_{\pi_*}(s, a) = \max_\pi Q_\pi(s, a)$, the Q-function can be estimated recursively using the Bellman optimality equation that is described by

$$Q_{\pi_*}(s, a) = \mathrm{E}[r_{t+1} + \gamma \max_{a_{t+1}} Q_{\pi_*}(s_{t+1}, a_{t+1}) | s, a] \tag{5}$$

where $Q_{\pi_*}(s, a)$ is the Q-value for choosing action a from state s following the optimal policy π_*, r_{t+1} is the immediate reward received from the environment, $Q_{\pi_*}(s_{t+1}, a_{t+1})$ is the Q-value for selecting action a_{t+1} from the next environment state s_{t+1}. Learning the Q-values for all possible state-action pairs is intractable for practical RL applications. Popular approaches approximate Q-function using deep convolutional neural networks (Lillicrap et al., 2015; Mnih et al., 2015, 2016; Silver et al., 2016).

In this work, we model the agent using an LSM, wherein the liquid-to-readout weights are trained to approximate the Q-function as described below. At any time instant t, we map the current environment state vector s_t to input neurons firing at a rate constrained between 0 and ϕ Hz over certain time period (denoted by T_{LSM}) following a Poisson process. The maximum Poisson firing rate ϕ is tuned to ensure sufficient input spiking activity for a given RL task. We follow the method outlined in Heeger (2000) to generate the Poisson spike trains as explained below. For a particular input neuron in the state vector, we first compute the probability of generating a spike at every LSM-simulation time-step based on the corresponding Poisson firing rate. Note that the time-steps in the RL task are orthogonal to the time-steps used for the numerical simulation of the liquid. Specifically, in-between successive time-steps t and $t + 1$ in the RL task, the liquid is simulated for a time period of T_{LSM} with 1ms separation between consecutive LSM-simulation time-steps. The probability of producing a spike at any LSM-simulation time-step is obtained by scaling the corresponding firing rate by 1,000. We generate a random number drawn from a uniform distribution between 0 and 1, and produce a spike if the random number is lesser than the neuronal spiking probability. At every LSM-simulation time-step, we feed the spike map of the current environment state and record the spiking outputs of the liquid-excitatory neurons. We accumulate the excitatory neuronal spikes and normalize the individual neuronal spike counts with the maximum possible spike count over the LSM-simulation period to obtain the high-dimensional representation (activation) of the environment state as discussed in the previous subsection 2.1. Note that the liquid state variables, such as the neuronal membrane potentials are not reset between successive RL time-steps so that some information of the past environment representations are still retained. The capability of the liquid to retain decaying memory of the past representations enables it to perform temporal integration over different RL time-steps such that the high-dimensional representation provided by the liquid for the current environment state also depends on decaying memory of the past environment representations. However, it is important to note that appropriate initialization of the LSM (detailed in subsection 2.1) is necessary to obtain useful high-dimensional representation for efficient training of the liquid-to-readout weights as experimentally validated in section 3.

The high-dimensional liquid activations are fed to the readout layer that is trained using backpropagation to approximate the Q-function by minimizing the mean square error between the Q-values predicted by the readout layer and the target Q-values following (Mnih et al., 2015) as described by the following equations:

$$\theta_{t+1} = \theta_t + \eta \left(Y_t - Q(s_t, a_t|\theta_t) \right) \nabla_{\theta_t} Q(s_t, a_t|\theta_t) \tag{6}$$

$$Y_t = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}|\theta_t) \tag{7}$$

where θ_{t+1} and θ_t are the updated and previous synaptic weights in the readout layer, respectively, η is learning rate, $Q(s_t, a_t|\theta_t)$ is vector representing the Q-values predicted by the readout layer for all possible actions given the current environment state s_t using the previous readout weights, $\nabla_{\theta_t} Q(s_t, a_t|\theta_t)$ is the gradient of the Q-values with respect to the readout weights, and Y_t is the vector containing the target Q-values that is obtained by feeding the next environment state s_{t+1} to the LSM while using the previous readout weights. To encourage exploration during training, we follow ϵ-greedy policy (Watkins, 1989) for selecting the actions based on the Q-values predicted by the LSM. Based on ϵ-greedy policy, we select a random action with probability ϵ and the optimal action, i.e., the action pertaining to the highest Q-value with probability $(1-\epsilon)$ during training. Initially, ϵ is set to a large value (closer to unity), thereby permitting the agent to pick a lot of random actions and effectively explore the environment. As training progresses, ϵ gradually decays to a small value, thereby allowing the agent to exploit its past experiences. During evaluation, we similarly follow ϵ-greedy policy albeit with much smaller ϵ so that there is a strong bias toward exploitation. Employing ϵ-greedy policy during evaluation also serves to mitigate the negative impact of over-fitting or under-fitting. In an effort to further improve stability during training and achieve better generalization performance, we use the experience replay technique proposed by Mnih et al. (2015). Based on experience replay, we store the experience discovered at each time-step (i.e., s_t, a_t, r_t, and s_{t+1}) in a large table and later train the LSM by sampling mini-batches of experiences in a random manner over multiple training epochs, leading to improved generalization performance. For all the experiments reported in this work, we use the RMSProp algorithm (Tieleman and Hinton, 2012) as the optimizer for error backpropagation with mini-batch size of 32. We adopt ϵ-greedy policy, wherein ϵ gradually decays from 1 to $0.001-0.1$ over the first 10% of the training steps. Replay memory stores one million recently played frames, which are then used for mini-batch weight updates that are carried out after the initial 100 training steps. The simulation hyperparameters for Q-learning are summarized in **Table 3**.

3. EXPERIMENTAL RESULTS

We first present results motivating the importance of careful LSM initialization for obtaining rich high-dimensional state representation, which is necessary for efficient training of the

TABLE 3 | Q-learning simulation parameters.

Parameter	Value
Readout weights update frequency	Once every game-step
Warm up steps before training begins	100
Batch size for experience replay	32
Experience replay buffer size	1×10^6
Discount factor	0.95
Initial exploration probability during training	1
Final exploration probability during training (Cartpole)	1×10^{-3}
Final exploration probability during training (Pacman & Atari)	1×10^{-1}
Exploration probability during evaluation (Cartpole & Atari)	5×10^{-2}
Exploration probability during evaluation (Pacman)	0
Learning rate for RMSProp algorithm	2×10^{-4}
Term added to denominator for RMSProp algorithm	1×10^{-6}
Weight decay for RMSProp algorithm	0
Smoothing constant for RMSProp algorithm	0.99

liquid-to-readout weights. We then demonstrate the utility of the recurrent-liquid synaptic connections of careful LSM initialization using classic cartpole-balancing RL task (Sutton and Barto, 1998). We then validate the capability of appropriately initialized LSM, trained using the presented methodology, for solving complex RL tasks like Pacman (DeNero et al., 2010) and Atari games (Brockman et al., 2016).

3.1. LSM Hyperparameter Tuning

Initializing LSM with appropriate hyperparameters is an important step to construct a model that produces useful high-dimensional representations. Since the input-to-liquid and recurrent-liquid connectivity matrices of the LSM are fixed *a priori* during training, how these connections are initialized dictates the liquid dynamics. We choose the hyperparameters K (governing the input-to-liquid connectivity matrix) and C (governing the recurrent-liquid connectivity matrices) empirically based on three observations: (1) stable spiking activity of the liquid, (2) eigenvalue analysis of the recurrent connectivity matrices, and (3) development of liquid-excitatory neuron membrane potential.

Spiking activity of the liquid is said to be stable if every finite stream of inputs results in a finite period of response. Sustained activity indicates that small input noise can perturb the liquid state and lead to chaotic activity that is no longer dependent on the input stimuli. It is impractical to analyze the stability of the liquid for all possible input streams within a finite time. We investigate the liquid stability by feeding in random input stimuli and sampling the excitatory neuronal spike counts at regular time intervals over the LSM-simulation period for different values of K and C. We separately adjust these hyperparameters for each learning task using random representations of the environment based on the following experimental steps. We begin by first selecting the hyper-parameter K, which indicates the number of pre-synaptic inputs to each neuron in the liquid. K is initialized to a small number (=1 in our experiments) while C is set to zero. We gradually increase K until the liquid neurons are sufficiently excited to determine the K that leads to optimally sparse spiking activity. The same optimal value of K can then be used for liquid of any size since each neuron still receives similar degree of excitation from the inputs and spikes sufficiently. Using the optimal value of K, we increase C until the desired eigenvalue spectrum and spiking neuronal dynamics (with respect to the evolution of the membrane potential over time) are obtained as explained in the following paragraph.

Analyzing the eigenvalue spectrum of the recurrent connectivity matrix is a common tool to assess the stability of the liquid. Each eigenvalue in the spectrum represents an individual mode of the liquid. Real part indicates decay rate of the mode while the imaginary part corresponds to the frequency of the mode (Rajan et al., 2010). Liquid spiking activity remains stable as long as all eigenvalues remain within the unit circle. However, this condition is not easily met for realistic recurrent-liquid connections with random synaptic weight initialization (Rajan and Abbott, 2006). We constrain the recurrent weights (hyperparameter β) such that each neuron receives balanced excitatory and inhibitory synaptic currents as previously discussed in subsection 2.1. This results in eigenvalues that lie within the unit circle as illustrated in **Figure 2A**. In order to emphasize the importance of LSM initialization, we also show the eigenvalue spectrum of the recurrent-liquid connectivity matrix when the weights are not properly initialized as shown in **Figure 2B** where many eigenvalues are outside the unit circle. Finally, we also use the development of the excitatory neuronal membrane potential to guide hyperparameter tuning. The hyperparameters C and β are chosen to ensure that membrane potential exhibits balanced fluctuation as illustrated in **Figure 2C** that plots the membrane potential of 10 randomly picked neurons in the liquid. Note that these steps to find K and C are based on empirical observations. We chose values of K and C to be 3 and 4 for cartpole and Pacman experiment, respectively, which ensures stable liquid spiking activity while enabling the liquid to exhibit fading memory of the past inputs.

3.2. Learning to Balance a Cartpole

Cartpole-balancing is a classic control problem wherein the agent has to balance a pole attached to a wheeled cart that can move freely on a rail of certain length as shown in **Figure 3A**. The agent can exert a unit force on the cart either to the left or right side for balancing the pole and keeping the cart within the rail. The environment state is characterized by cart position, cart velocity, angle of the pole, and angular velocity of the pole, which are designated by the tuple $(\chi, \dot{\chi}, \varphi, \dot{\varphi})$. The environment returns a unit reward every time-step and concludes after 200 time-steps if the pole does not fall or the cart does not goes out of the rail. Because the game is played for a finite time period, we constrain $(\chi, \dot{\chi}, \varphi, \dot{\varphi})$ to be within the range specified by $(\pm 2.5, \pm 0.5, \pm 0.28, \pm 0.88)$ for efficiently mapping the real-valued state inputs to spike trains feeding into the LSM. Each real-valued state input is mapped to 10 input neurons which have firing rates proportional to one-hot encoding of the input value representing 10 distinct levels within the corresponding range.

We model the agent using an LSM containing 150 liquid neurons, 32 hidden neurons in the fully-connected layer between the liquid and output layer, and two output neurons. The maximum firing rate for the input neurons representing the environment state is set to 100 Hz and each input is presented for 100 ms. The LSM is trained for 10^5 time-steps, which are equally divided into 100 training epochs containing 1,000 time-steps per epoch. After each epoch, the LSM is evaluated for 1,000 time-steps with the probability of choosing a random action ϵ set to 0.05. Note that the LSM is evaluated for 1,000 time-steps (multiple gameplays) even though single gameplay lasts a maximum of only 200 time-steps as mentioned in the previous paragraph. We use the accumulated reward averaged over multiple gameplays as the true indicator of the LSM (agent) performance to account for the randomness in action-selection as a result of the ϵ-greedy policy. We train the LSM initialized with 10 different random seeds and obtain median accumulated reward as shown in **Figure 3B**. Note that the maximum possible accumulated reward per gameplay is 200 since each gameplay lasts at most 200 time-steps. Increase in median accumulated reward over epochs indicates that the LSM learnt to balance the cartpole using the dynamically

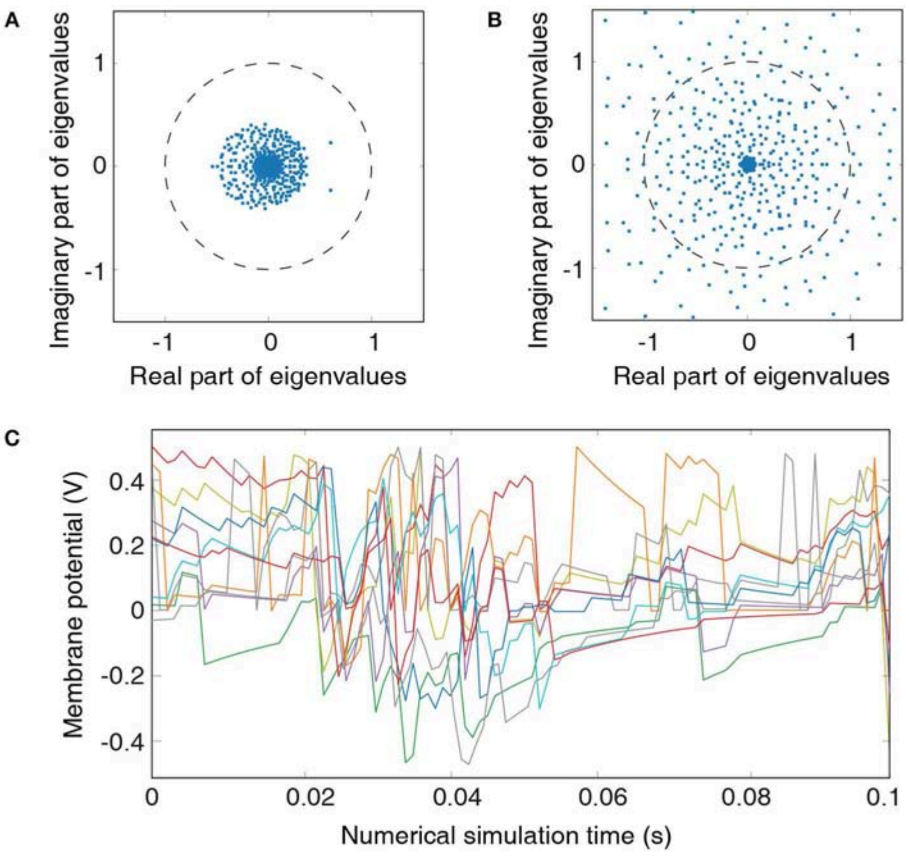


FIGURE 2 | Metrics for guiding hyperparameter tuning: **(A)** Eigenvalue spectrum of the recurrent-liquid connectivity matrix for an LSM containing 500 liquid neurons. The LSM is initialized with synaptic weights listed in **Table 1** based on hyperparameter *C*=4. All eigenvalues in the spectrum lie within a unit circle. **(B)** Eigenvalue spectrum of the recurrent-liquid connectivity matrix initialized with synaptic weights $\beta_{E\to E} = 0.4$, $\beta_{E\to I} = 0.1$, and $\beta_{I\to E} = 0.1$. Many eigenvalues in the spectrum are outside the unit circle. **(C)** Development of membrane potentials from 10 randomly picked excitatory neurons in the liquid initialized with synaptic weights listed in **Table 1** based on hyperparameter $C = 4$. Random representation from the cartpole-balancing problem is used as the input.

evolving high-dimensional liquid states. The ability of the liquid to provide rich high-dimensional input representations can be attributed to the careful initialization of the connectivity matrices and weights (explained in subsection 2.1), which ensures balance between the excitatory and inhibitory currents to the liquid neurons and preserves fading memory of past liquid activity. However, the median accumulated reward after 100 training epochs saturates around 125 and does not reach the maximum value of 200. We hypothesize that the game score saturation comes from the quantized representation of the environment state, and demonstrate in the following experiment with Pacman that the LSM can learn optimally given a better state representation. Finally, in order to emphasize the importance of LSM initialization, we also show the median accumulated reward per training epoch for training in which the LSM is initialized to have few synaptic connections. **Figure 3C** indicates that the median accumulated reward is around 90 when the LSM initialization is suboptimal.

To visualize the learnt action-value function guiding action selection, we compare Q-values produced by the LSM during evaluation in three different scenarios depicted in **Figure 3D**. Note that each Q-value represents how good is the corresponding action for a given environment state. In scenario 1 (see **Figure 3D-1**) that corresponds to the beginning of the gameplay wherein the pole is almost balanced, the value of both the actions are identical. This implies that either action (moving the cart left or right) will lead to a similar outcome. In scenario 2 (see **Figure 3D-2**) wherein the pole is unbalanced to the left side, the difference between the predicted Q values increases. Specifically, the Q value for applying a unit force on the right side of the cart is higher, which causes the cart to move to the left. Pushing the cart to the left in turn causes the pole to swing back right toward the balanced position. Similarly, in scenario 3 (see **Figure 3D-3**) wherein the pole is unbalanced to the right side, the Q value is higher for applying a unit force on the left side of the cart, which causes the cart to move right and enables the pole to swing left toward the balanced position. This visually demonstrates the ability of the LSM (agent) to successfully balance the pole by pushing the cart appropriately to the left or right based on the learnt Q values.

3.3. Learning to Balance a Cartpole Without Complete State Information

In this sub-section, we demonstrate the capability of the LSM to learn without complete state information, thereby validating its ability to perform temporal integration across different RL game steps enabled by the sparse random recurrent connections. Specifically, we modify the previous cartpole-balancing task such that the agent only receives the cart position and angle of the pole, designated by tuple (χ, φ), as an input while the velocity information is ignored. The objective is to determine if the decaying memory of the past cart position and pole angle retained by the liquid, as a result of the recurrent-liquid connectivity, enables the LSM to make better decisions without the velocity information. We clip (χ, φ) to be within the range specified by $(\pm 2.5, \pm 0.28)$ similar to the previous experiment; however, each real-valued state input is mapped to only 1 input neuron whose firing rate is proportional to the normalized state value. A positive state input causes the corresponding neuron to fire unit positive spikes. On the other hand, if the state input is negative, the input neuron fires unit negative spikes at a rate proportional to the absolute value of the input as described in Sengupta et al. (2019). We initialize the input-to-liquid weights from a uniform distribution between -0.4 and 0.4 to achieve balanced input excitation in the presence of both positive and negative spikes. Other connection weights are initialized from a uniform distribution as shown in **Table 4**.

We model the agent using an LSM with 150 liquid neurons followed by a fully-connected layer with 32 hidden neurons and a final output layer with two neurons, which is similar to the architecture used for the previous cartpole-balancing experiment. Additional feedback connections between excitatory neurons that have a large delay of 20 ms are introduced to

TABLE 4 | Synaptic weight initialization parameters for learning to balance cartpole without complete state information.

Connection type	Weight
RECURRENT-LIQUID CONNECTIONS	
$E \rightarrow E$ with 1 ms delay	[0, 0.4]
$E \rightarrow E$ with 20 ms delay	[0, 0.4]
$E \rightarrow I$	[0, 0.4]
$I \rightarrow E$	[0, 0.4]
$I \rightarrow I$	[0, 0.01]

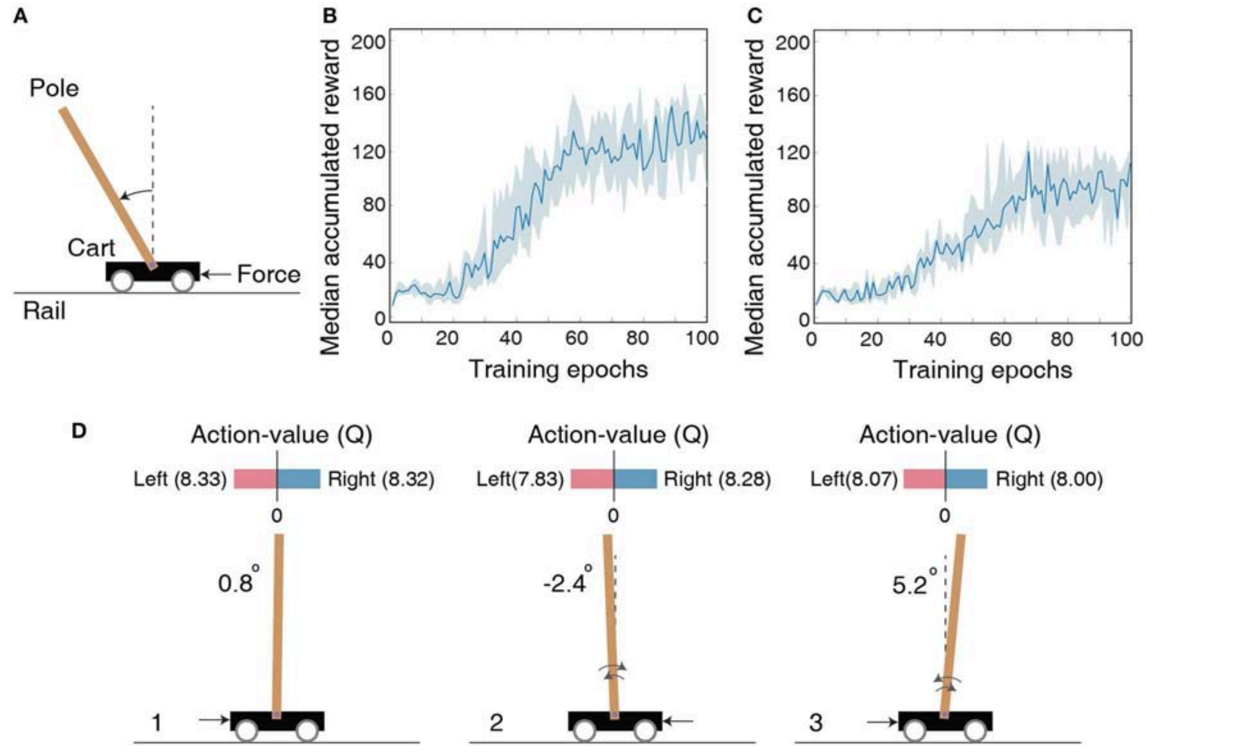


FIGURE 3 | (A) Illustration of the cartpole-balancing task wherein the agent has to balance a pole attached to a wheeled cart that moves freely on a rail of certain length. **(B)** The median accumulated reward per epoch provided by the LSM trained across 10 different random seeds for the cartpole-balancing task. Shaded region in the plot represents the 25-th to 75-th percentile of the accumulated reward over multiple random seeds. **(C)** The median accumulated reward per epoch from cartpole training across 10 different random seeds in which the LSM is initialized to have sparser connectivity between the liquid neurons compared to that used for the experiment in **(B)**. **(D)** Visualization of the learnt Q (action-value) function for the cartpole-balancing task at three different game-steps designated as 1, 2, and 3. Angle of the pole is written on the left side of each figure. Negative angle represents an unbalanced pole to the left and positive angle represents an unbalanced pole to the right. Black arrow corresponds to a unit force on the left or right side of the cart depending on which Q value is larger.

achieve long-term temporal integration over RL time-steps. In this experiment, we also reduced the LSM simulation time-steps to 20 ms from 100 ms used in the previous experiment to precisely validate the long-term temporal integration capability of the liquid. The LSM is trained for a total of 5×10^6 time-steps, which is sufficiently long to guarantee no further improvement in performance. Without complete state information, the LSM achieves best median accumulated reward of 70.93 over the last 10 epochs as illustrated in **Figure 4**, which is lower than that (125) attained with complete state information. However, the median accumulated reward of 70.93 achieved by the LSM based on incomplete state information is still higher than that (38.23) provided by the LSM without recurrent connections as shown in **Figure 4**. This indicates that the sparse recurrent connections provide useful information about the past input since information about the cart velocity and angular velocity of the pole can be derived based on the current and past cart position and pole angle. We observe that LSM initialized based on some random seeds provide significantly better learning than others due to inherent stochasticity in the model, but we report the results based on the reward statistics obtained using runs from 5 different random seeds.

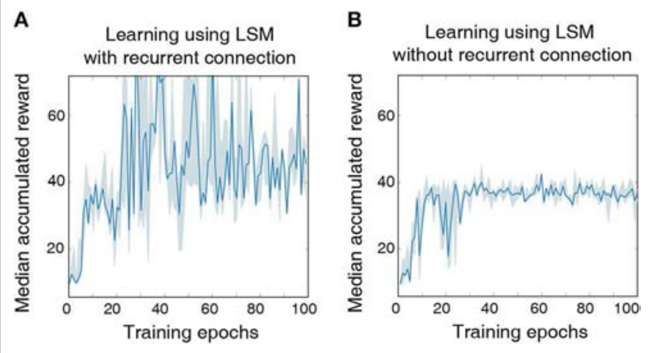


FIGURE 4 | (A) The median accumulated reward per epoch obtained from cartpole training with five different random seeds using an LSM with sparse random recurrent connections. **(B)** The median accumulated reward per epoch obtained from cartpole training across the same five different random seeds using an LSM without any recurrent connections. Shaded region in the plot represents the 25-th to 75-th percentile of the accumulated reward over multiple random seeds.

3.4. Learning to Play Pacman

In order to comprehensively validate the efficacy of the high-dimensional environment representations provided by the liquid, we train the LSM to play a game of Pacman (DeNero et al., 2010). The objective of the game is to control Pacman (yellow in color) to capture all the foods (represented by small white dots) in a grid without being eaten by the ghosts as illustrated in **Figure 5**. The ghosts always hunt the Pacman; however, cherry (represented by large white dots) make the ghosts temporarily scared of the Pacman and run away. The game environment returns unit reward whenever Pacman consumes food, cherry, or the scared ghost (white in color). The game environment also returns a unit reward and restarts when all foods are captured. We use the location of Pacman, food, cherry, ghost and scared ghost as the environment state representation. The location of each object is encoded as a two-dimensional binary array whose dimension matches with that of the Pacman grid as shown in **Figure 5**. The binary intermediate representations of all the objects are then concatenated and flattened into a single vector to be fed to the input layer of the LSM.

The LSM configurations and game settings used for Pacman experiments are summarized in **Table 5**, where each game setting has different degree of complexity with regards to the Pacman grid size and the number of foods, ghosts, and cherries. In the first experiment, we use a 7×7 grid with three foods for Pacman to capture and a single ghost to prevent it from achieving its objective. Thus, the maximum possible accumulated reward at the end of a successful game is 4. **Figure 6A** shows that the

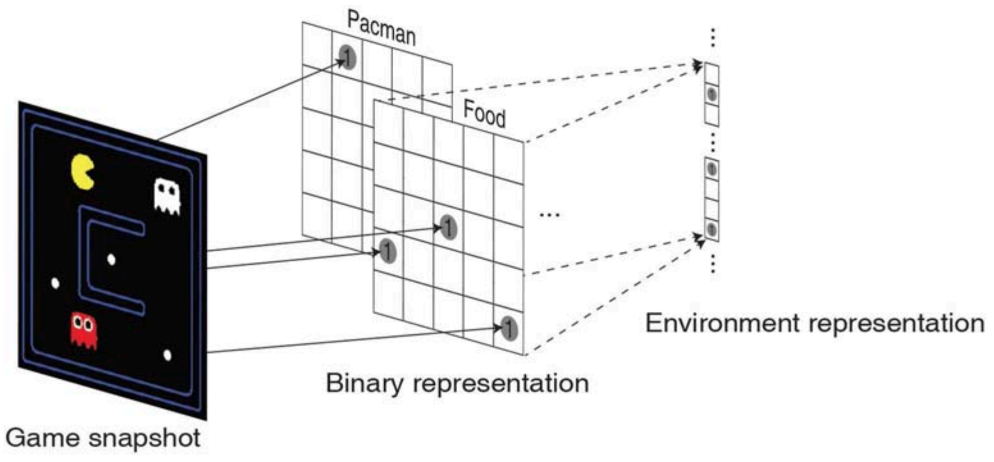


FIGURE 5 | Illustration of a snapshot from the Pacman game that is translated into 5 two-dimensional binary representations corresponding to the location of Pacman, foods, cherries, ghosts, and scared ghosts. The binary intermediate representations are then flattened and concatenated to obtain the environment state representation.

median accumulated reward gradually increases with the number of training epochs and converges closer to the maximum possible reward, thereby validating the capability of the liquid to provide useful high-dimensional representation of the environment state necessary for efficient training of the readout weights using the presented methodology. Interestingly, in the second experiment using a larger 7 × 17 grid, we find that the median reward converges to 12, which is greater than the number of foods available in the grid. This indicates that the LSM does not only learn to capture all the foods; in addition, it also learns to capture the cherry and the scared ghosts, leading to further increase the accumulated reward since consuming the scared ghost results in a unit immediate reward. In the final experiment, we train the LSM to control Pacman in 17 × 19 grid with sparsely dispersed foods. We find that larger grid requires more exploration and training steps for the agent to perform well and achieve the maximum possible reward, resulting in a learning curve that is less steep compared to that obtained for smaller grid sizes in the earlier experiments as shown in **Figure 6C**.

Finally, we plot the average of Q-values produced by the LSM as the Pacman navigates the grid to visualize the correspondence between the learnt Q-values and the environment state. As discussed in subsection 2.2, each Q-value produced by the LSM provides a measure of how good is a particular action for a give environment state. The Q-value averaged over the set of all possible actions (known as the state-value function) thus indicates the value of being in a certain state. **Figure 6D** illustrates the state-value function while playing the Pacman game in a 7×17 grid. The predicted state-value starts at a relatively high level because the foods are abundant in the grid and the ghosts are far away from the Pacman (see **Figure 6D-1**). The state-value gradually decreases as the Pacman navigates through the grid and gets closer to the ghosts. The predicted state-value then shoots up after the Pacman consumes cherry and makes the ghosts temporarily consumable (see **Figure 6D-2**), leading to potential additional reward. The predicted state-value drops after the ghosts are reborn (see **Figure 6D-3**). Finally, we observe a slight increase in the state-value toward the end of the game when the Pacman is closer to the last food after it consumes a cherry (see **Figure 6D**). It is interesting to note

TABLE 5 | LSM configuration and game settings for different Pacman experiments reported in this work.

Grid size	Ghost	Food	Cherry	Training steps	Liquid neurons	Hidden neurons
7×7	1	3	0	5×10^5	500	128
7×17	2	6	2	5×10^5	2,000	512
17×19	1	6	0	3×10^6	3,000	512

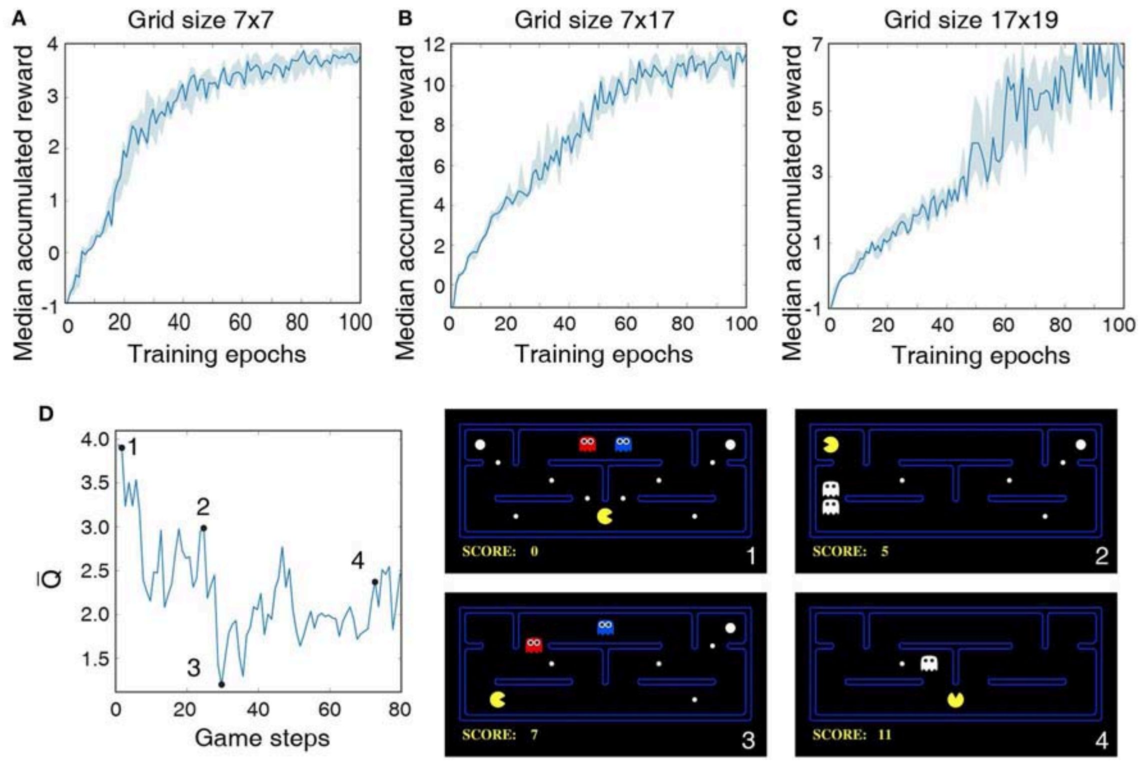


FIGURE 6 | Median accumulated reward per epoch obtained by training and evaluating the LSM on three different game settings: **(A)** grid size 7 × 7, **(B)** grid size 7 × 17, and **(C)** grid size 17 × 19. LSM is initialized and trained with 7 different initial random seeds. Shaded region represents the 25-th to 75-th percentile of the accumulated reward over multiple seeds. **(D)** The plot on the left shows the predicted state-value function for 80 continuous Pacman game steps. The four snapshots from the Pacman game shown on the right correspond to game steps designated as 1, 2, 3, and 4, respectively, in the state-value plot.

that although the scenario in **Figure 6D-4** is similar to that in **Figure 6D-2**, the state-value is smaller since the expected accumulated reward at this step is at most 3 assuming that the Pacman can capture both the scared ghost and the last food. On the other hand, in the environment state shown in **Figure 6D-2**, the expected accumulated reward is >3 since 4 foods and 2 scared ghosts are available for the Pacman to capture.

3.5. Learning to Play Atari Games

Finally, we train the LSM using the presented methodology to play Atari games (Brockman et al., 2016), which are widely used to benchmark deep reinforcement learning networks. We arbitrarily select 4 games for evaluation, namely, Boxing, Gopher, Freeway, and Krull. We use the RAM of the Atari machine, which stores 128 bytes of information about an Atari game, as a representation of the environment (Brockman et al., 2016). During training, we modified the reward structure of the game by clipping all positive immediate rewards to $+1$ and all negative immediate rewards to -1. However, we do not clip the immediate reward during testing and measure the actual accumulated reward following Mnih et al. (2015). For all selected Atari games, we model the agent using an LSM containing 500 liquid neurons and 128 hidden neurons. Number of output neurons varies for each game as the number of possible actions is different. The maximum Poisson firing rate for the input neurons is set to 100 Hz and each input is presented for 100ms. The LSM is trained for 5×10^3 steps.

Figure 7 illustrates that the LSM successfully learnt to play the Atari games without any prior knowledge of the rules, leading to gradually increasing accumulated reward with the number of training epochs. We compare the median accumulated reward provided by the LSM to the average accumulated reward obtained from playing with random actions for 1×10^5 steps. Note that the median accumulated reward used for comparison is the highest reward achieved during the evaluation phase over the last 10 training epochs. **Table 6** shows that the median accumulated reward offered by the LSM is higher than the average accumulated reward obtained with random actions for all the four Atari games, which demonstrates the capability of the LSM to learn successful strategies in complex RL tasks. In fact, the median accumulated reward on Boxing and Krull reach the same level as human players reported in Mnih et al. (2015). However, we observe that the median accumulated reward on Freeway and Gopher are much lower than that of the human players. In order to identify the cause of poor learning, we trained all selected games using a deep learning network consisting of two convolutional and two fully-connected layers, and compared the median accumulated reward with that provided by the LSM. The architecture of the deep learning network used for different games is listed in **Table 7**. **Table 6** shows that the deep learning network trained with end-to-end error backpropagation using the Q-learning algorithm achieves better than human-level performance on Boxing and Krull while yielding lower rewards on Freeway and Gopher. Hence, the inferior performance of the LSM on Freeway and Gopher can be attributed to the nature (or complexity) of the respective games. However, the deep learning network yields superior performance compared to that provided by the LSM on all selected Atari games. We believe that the gap in the LSM performance compared to that obtained using the deep learning network stems from the inability of a randomly initialized LSM to extract complex input representations and game strategies. On the computation perspective, training a deep learning network incurs higher cost due to additional trainable parameters and the need for carrying out end-to-end error backpropagation. Simpler models like LSMs with lower training complexity offer a possible alternative for efficient training and inference in edge devices, such as self-flying drones that

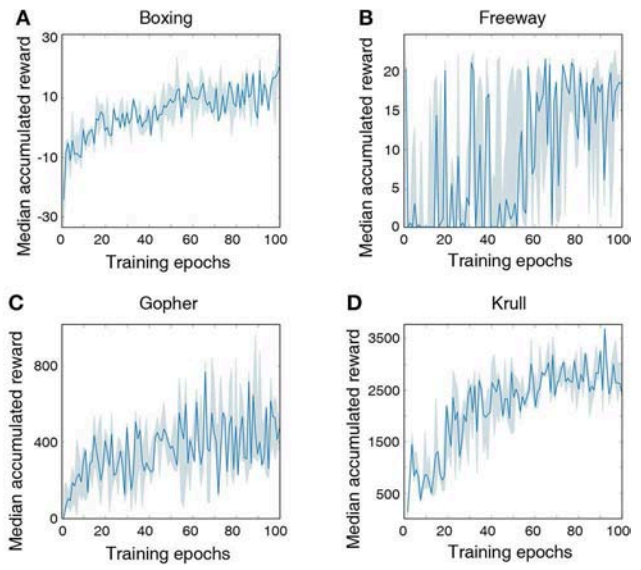


FIGURE 7 | Median accumulated reward per epoch obtained by training and evaluating the LSM on 4 selected Atari games: **(A)** Boxing, **(B)** Freeway, **(C)** Gopher, and **(D)** Krull. For each game, LSM is initialized and trained with five different initial random seeds. Shaded region represents the 25-th to 75-th percentile of the accumulated reward over multiple seeds.

TABLE 6 | Median accumulated reward for each game is chosen from the highest median accumulated reward over the last 10 training epochs across five different initial random seeds.

Game	Learning with LSM	Random actions	Human player	Learning with deep network
Boxing	20.2	0.8	4.3	68.2
Freeway	19.8	0.0	29.6	21.6
Gopher	611.1	279.3	2,321	1,443
Krull	3,686	1,590	2,395	4,672

Columns 1 and 4 report median accumulated rewards from learning with LSM and deep network, respectively. Average accumulated reward in column 2 is obtained from playing with random actions for 1×10^5 steps, which is a sufficiently large number for the average accumulated reward to be stable. Accumulated reward from human players reported in Mnih et al. (2015) is listed in column 3 for every game.

TABLE 7 | Convolutional deep learning network architecture used in Atari experiments.

Layer	Output features	Kernal size	Stride	Padding
One-dimensional convolutional	4	4	2	1
One-dimensional convolutional	16	4	2	1
Fully connected	128			
Fully connected	3 for Freeway 8 for Gopher 18 for Boxing and Krull			

operate under computational resource constraints and limited power budget.

4. DISCUSSION

LSM, an important class of biologically plausible recurrent SNNs, has thus far been primarily demonstrated for pattern (speech/image) recognition (Bellec et al., 2018; Srinivasan et al., 2018), gesture recognition (Chrol-Cannon and Jin, 2015; Panda and Srinivasa, 2018), and sequence generation tasks (Nicola and Clopath, 2017; Panda and Roy, 2017; Bellec et al., 2019) using standard datasets. To the best of our knowledge, our work is the first demonstration of LSMs, trained using Q-learning based methodology, for complex RL tasks like Pacman and Atari games commonly used to evaluate deep reinforcement learning networks. The benefits of the proposed LSM-based RL framework over the state-of-the-art deep learning models are 2-fold. First, LSM entails fewer trainable parameters as a result of using fixed input-to-liquid and recurrent-liquid synaptic connections. However, this requires careful initialization of the respective matrices for efficient training of the liquid-to-readout weights as experimentally validated in section 3. We note that the performance of LSMs could be further improved by training the recurrent weights using localized Spike Timing Dependent Plasticity (STDP) based learning rules (Bi and Poo, 1998; Song et al., 2000; Diehl and Cook, 2015) as demonstrated in Panda and Roy (2017) or biologically inspired variants of backpropagation-through-time (Bellec et al., 2018, 2019). Second, LSMs can be efficiently implemented on event-driven neuromorphic hardware like IBM *TrueNorth* (Merolla et al., 2014) or Intel *Loihi* (Davies et al., 2018), leading to potentially much improved energy efficiency while achieving comparable performance to deep learning models on the chosen benchmark tasks. Note that the readout layer in the presented LSM needs to be implemented outside the neuromorphic fabric since they are composed of artificial rate-based neurons that are typically not supported in neuromorphic hardware realizations. Alternatively, readout layer composed of spiking neurons could be used that can be trained using spike-based error backpropagation algorithms (Lee et al., 2016, 2018; Panda and Roy, 2016; Jin et al., 2018; Wu et al., 2018; Bellec et al., 2019). Future works could also explore STDP-based reinforcement learning rules (Pfister et al., 2006; Farries and Fairhall, 2007; Florian, 2007; Legenstein et al., 2008) to render the training algorithm amenable for neuromorphic hardware implementations.

5. CONCLUSION

Liquid State Machine (LSM) is a bio-inspired recurrent spiking neural network composed of an input layer sparsely connected to a randomly interlinked liquid of spiking neurons for the real-time processing of spatio-temporal inputs. In this work, we proposed LSMs, trained using Q-learning based methodology, for solving complex Reinforcement Learning (RL) tasks like playing Pacman and Atari that have been hitherto benchmarked for deep reinforcement learning networks. We presented initialization strategies for the fixed input-to-liquid and recurrent-liquid connectivity matrices and weights to enable the liquid to produce useful high-dimensional representation of the environment based on the current and past input states necessary for efficient training of the liquid-to-readout weights. We demonstrated the significance of the sparse recurrent connections, which enables the liquid to retain decaying memory of the past input representations and perform temporal integration across RL time-steps, by training it using partial input state information that yielded higher accumulated reward than that provided by a liquid without recurrent connections. Our experiments on the Pacman game showed that the LSM learns the optimal strategies for different game settings and grid sizes. Our analyses on a subset of Atari games indicated that the LSM achieves comparable score to that reported for human players in existing works.

DATA AVAILABILITY

Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/openai/gym.

AUTHOR CONTRIBUTIONS

GS and WP wrote the paper. WP performed the simulations. All authors helped with developing the concepts, conceiving the experiments, and writing the paper.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by the Semiconductor Research Corporation, the National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship, and by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

REFERENCES

Amit, D. J. (1992). *Modeling Brain Function: The World of Attractor Neural Networks*. New York, NY: Cambridge University Press.

Auer, P., Burgsteiner, H., and Maass, W. (2002). "Reducing communication for distributed learning in neural networks," in *International Conference on Artificial Neural Networks* (Madrid: Springer), 123–128.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems 2018* (Quebec). Available online at: http://papers.nips.cc/paper/7359-long-short-term-memory-and-learning-to-learn-in-networks-of-spiking-neurons

Bellec, G., Scherr, F., Hajek, E., Salaj, D., Legenstein, R., and Maass, W. (2019). Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv [Preprint] arXiv:1901.09049*. Available online at: https://arxiv.org/abs/1901.09049

Berberich, N. (2017). *Implementation of a real-time liquid state machine on spinnaker for biomimetic robot controll (Masterarbeit)*. Munich: TUM.

Bi, G.-q., and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). Openai gym. *arXiv [Preprint] arXiv:1606.01540*. Available online at: https://arxiv.org/abs/1606.01540

Chrol-Cannon, J., and Jin, Y. (2015). Learning structure of sensory inputs with synaptic plasticity leads to interference. *Front. Comput. Neurosci.* 9:103. doi: 10.3389/fncom.2015.00103

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. (2003). Theoretical neuroscience: computational and mathematical modeling of neural systems. *J. Cognit. Neurosci.* 15, 154–155. doi: 10.1162/089892903321107891

DeNero, J., and Klein, D. (2010). "Teaching introductory artificial intelligence with pac-man," in *First AAAI Symposium on Educational Advances in Artificial Intelligence* (California), 1885–1889.

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Douglas, R. J., Koch, C., Mahowald, M., Martin, K., and Suarez, H. H. (1995). Recurrent excitation in neocortical circuits. *Science* 269, 981–985. doi: 10.1126/science.7638624

Farries, M. A., and Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing–dependent synaptic plasticity. *J. Neurophysiol.* 98, 3648–3665. doi: 10.1152/jn.00364.2007

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Comput.* 19, 1468–1502. doi: 10.1162/neco.2007.19.6.1468

Harris, K. D., and Mrsic-Flogel, T. D. (2013). Cortical connectivity and sensory coding. *Nature* 503:51. doi: 10.1038/nature12654

Heeger, D. (2000). Poisson model of spike generation. *Handout* 5, 1–13. Available online at: https://www.cns.nyu.edu/~david/handouts/poisson.pdf

Jiang, X., Shen, S., Cadwell, C. R., Berens, P., Sinz, F., Ecker, A. S., et al. (2015). Principles of connectivity among morphologically defined cell types in adult neocortex. *Science* 350:aac9462. doi: 10.1126/science.aac9462

Jin, Y., Zhang, W., and Li, P. (2018). "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems,* eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Montréal, QC: Curran Associates), 7005–7015.

Joshi, P., and Maass, W. (2005). Movement generation with circuits of spiking neurons. *Neural Comput.* 17, 1715–1738. doi: 10.1162/0899766054026684

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Legenstein, R., Pecevski, D., and Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput. Biol.* 4:e1000180. doi: 10.1371/journal.pcbi.1000180

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations 2016* (San Juan, PR). Available online at: https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:main.html

Lukoševičius, M., and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.* 3, 127–149. doi: 10.1016/j.cosrev.2009.03.005

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Maass, W., Natschläger, T., and Markram, H. (2003). "A model for real-time computation in generic neural microcircuits," in *Advances in Neural Information Processing Systems,* eds S. Becker, S. Thrun, and K. Obermayer (Vancouver, BC: Curran Associates), 229–236.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning* (New York, NY), 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518:529. doi: 10.1038/nature14236

Nair, V., and Hinton, G. E. (2010). "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Haifa), 807–814.

Nicola, W., and Clopath, C. (2017). Supervised learning in spiking neural networks with force training. *Nat. Commun.* 8:2208. doi: 10.1038/s41467-017-01827-3

Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 299–306.

Panda, P., and Roy, K. (2017). Learning to generate sequences with combination of hebbian and non-hebbian plasticity in recurrent spiking neural networks. *Front. Neurosci.* 11:693. doi: 10.3389/fnins.2017.00693

Panda, P., and Srinivasa, N. (2018). Learning to recognize actions from limited training examples using a recurrent spiking neural model. *Front. Neurosci.* 12:126. doi: 10.3389/fnins.2018.00126

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Comput.* 18, 1318–1348. doi: 10.1162/neco.2006.18.6.1318

Rajan, K., and Abbott, L. (2006). Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* 97:188104. doi: 10.1103/PhysRevLett.97.188104

Rajan, K., Abbott, L., and Sompolinsky, H. (2010). Stimulus-dependent suppression of chaos in recurrent neural networks. *Phys. Rev. E* 82:011903. doi: 10.1103/PhysRevE.82.011903

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323:533. doi: 10.1038/323533a0

Savage, J. E. (1998). *Models of Computation*, Vol. 136. Reading, MA: Addison-Wesley.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: Vgg and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484. doi: 10.1038/nature16961

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3:919. doi: 10.1038/78829

Srinivasan, G., Panda, P., and Roy, K. (2018). Spilinc: spiking liquid-ensemble computing for unsupervised speech and image recognition. *Front. Neurosci.* 12:524. doi: 10.3389/fnins.2018.00524

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tieck, J. C. V., Pogančić, M. V., Kaiser, J., Roennau, A., Gewaltig, M.-O., and Dillmann, R. (2018). "Learning continuous muscle control for a multi-joint arm by extending proximal policy optimization with a liquid state machine," in *International Conference on Artificial Neural Networks* (Rhodes: Springer), 211–221.

Tieleman, T., and Hinton, G. (2012). Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA Neural Netw. Mach. Learn.* 4, 26–31. Available online at: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Verstraeten, D., Schrauwen, B., Stroobandt, D., and Van Campenhout, J. (2005). Isolated word recognition with the liquid state machine: a case study. *Inform. Process. Lett.* 95, 521–528. doi: 10.1016/j.ipl.2005.05.019

Watkins, C. J., and Dayan, P. (1992). Q-learning. *Mach. Learn.* 8, 279–292. doi: 10.1023/A:1022676722315

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* (PhD thesis), King's College, Cambridge, United Kingdom.

Wehr, M., and Zador, A. M. (2003). Balanced inhibition underlies tuning and sharpens spike timing in auditory cortex. *Nature* 426:442. doi: 10.1038/nature02116

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* 12:331. doi: 10.3389/fnins.2018.00331

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.