

The background of the entire page features a stylized brain composed of various colored segments (yellow, orange, red, purple, blue, green) arranged in a circular pattern. Overlaid on this brain is a network of white lines connecting small white dots, representing a neural or computational network. The top half of the image has a solid blue background, while the bottom half is white.

UNDERSTANDING AND BRIDGING THE GAP BETWEEN NEUROMORPHIC COMPUTING AND MACHINE LEARNING

EDITED BY: Lei Deng, Kaushik Roy and Huajin Tang

PUBLISHED IN: Frontiers in Computational Neuroscience and
Frontiers in Neuroscience



frontiers

Frontiers eBook Copyright Statement

The copyright in the text of individual articles in this eBook is the property of their respective authors or their respective institutions or funders. The copyright in graphics and images within each article may be subject to copyright of other parties. In both cases this is subject to a license granted to Frontiers.

The compilation of articles constituting this eBook is the property of Frontiers.

Each article within this eBook, and the eBook itself, are published under the most recent version of the Creative Commons CC-BY licence.

The version current at the date of publication of this eBook is CC-BY 4.0. If the CC-BY licence is updated, the licence granted by Frontiers is automatically updated to the new version.

When exercising any right under the CC-BY licence, Frontiers must be attributed as the original publisher of the article or eBook, as applicable.

Authors have the responsibility of ensuring that any graphics or other materials which are the property of others may be included in the CC-BY licence, but this should be checked before relying on the CC-BY licence to reproduce those materials. Any copyright notices relating to those materials must be complied with.

Copyright and source acknowledgement notices may not be removed and must be displayed in any copy, derivative work or partial copy which includes the elements in question.

All copyright, and all rights therein, are protected by national and international copyright laws. The above represents a summary only. For further information please read Frontiers' Conditions for Website Use and Copyright Statement, and the applicable CC-BY licence.

ISSN 1664-8714

ISBN 978-2-88966-742-0

DOI 10.3389/978-2-88966-742-0

About Frontiers

Frontiers is more than just an open-access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers Journal Series

The Frontiers Journal Series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the Frontiers Journal Series operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to Quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews.

Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the Frontiers Journals Series: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area! Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers Editorial Office: frontiersin.org/about/contact

UNDERSTANDING AND BRIDGING THE GAP BETWEEN NEUROMORPHIC COMPUTING AND MACHINE LEARNING

Topic Editors:

Lei Deng, Tsinghua University, China

Kaushik Roy, Purdue University, United States

Huajin Tang, Zhejiang University, China

Citation: Deng, L., Roy, K., Tang, H., eds. (2021). Understanding and Bridging the Gap between Neuromorphic Computing and Machine Learning. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-88966-742-0

Table of Contents

05	<i>Editorial: Understanding and Bridging the Gap Between Neuromorphic Computing and Machine Learning</i>
	Lei Deng, Huajin Tang and Kaushik Roy
09	<i>A Curiosity-Based Learning Method for Spiking Neural Networks</i>
	Mengting Shi, Tielin Zhang and Yi Zeng
21	<i>Corrigendum: A Curiosity-Based Learning Method for Spiking Neural Networks</i>
	Mengting Shi, Tielin Zhang and Yi Zeng
22	<i>Probabilistic Circuits for Autonomous Learning: A Simulation Study</i>
	Jan Kaiser, Rafatul Faria, Kerem Y. Camsari and Supriyo Datta
29	<i>Spike-Train Level Direct Feedback Alignment: Sidestepping Backpropagation for On-Chip Training of Spiking Neural Nets</i>
	Jeongjun Lee, Renqian Zhang, Wenrui Zhang, Yu Liu and Peng Li
40	<i>Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition</i>
	Jibin Wu, Emre Yilmaz, Malu Zhang, Haizhou Li and Kay Chen Tan
54	<i>Noise Helps Optimization Escape From Saddle Points in the Synaptic Plasticity</i>
	Ying Fang, Zhaofei Yu and Feng Chen
71	<i>Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks</i>
	Alexander Kugele, Thomas Pfeil, Michael Pfeiffer and Elisabetta Chicca
84	<i>Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)</i>
	Jacques Kaiser, Hesham Mostafa and Emre Neftci
95	<i>Neuromorphic Systems Design by Matching Inductive Biases to Hardware Constraints</i>
	Lorenz K. Muller, Pascal Stark, Bert Jan Offrein and Stefan Abel
106	<i>Supervised Learning in All FeFET-Based Spiking Neural Network: Opportunities and Challenges</i>
	Sourav Dutta, Clemens Schafer, Jorge Gomez, Kai Ni, Siddharth Joshi and Suman Datta
120	<i>Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization</i>
	Priyadarshini Panda, Sai Aparna Aketi and Kaushik Roy
138	<i>Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design</i>
	Maryam Parsa, John P. Mitchell, Catherine D. Schuman, Robert M. Patton, Thomas E. Potok and Kaushik Roy
154	<i>Exploring Neuromodulation for Dynamic Learning</i>
	Anurag Daram, Angel Yanguas-Gil and Dhireesha Kudithipudi

169 Bitstream-Based Neural Network for Scalable, Efficient, and Accurate Deep Learning Hardware

Hyeonuk Sim and Jongeun Lee

186 End-to-End Implementation of Various Hybrid Neural Networks on a Cross-Paradigm Neuromorphic Chip

Guanrui Wang, Songchen Ma, Yujie Wu, Jing Pei, Rong Zhao and Luping Shi



Editorial: Understanding and Bridging the Gap Between Neuromorphic Computing and Machine Learning

Lei Deng^{1*}, Huajin Tang^{2,3} and Kaushik Roy⁴

¹ Department of Precision Instrument, Center for Brain Inspired Computing Research (CBICR), Tsinghua University, Beijing, China, ² College of Computer Science and Technology, Zhejiang University, Hangzhou, China, ³ Zhejiang Lab, Hangzhou, China, ⁴ Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Keywords: neuromorphic computing, spiking neural networks, machine learning, artificial neural networks, cross paradigm

Editorial on the Research Topic

Understanding and Bridging the Gap between Neuromorphic Computing and Machine Learning

INTRODUCTION

On the road toward artificial general intelligence (AGI), two solution paths have been explored: neuroscience-driven neuromorphic computing such as spiking neural networks (SNNs) and computer-science-driven machine learning such as artificial neural networks (ANNs). Owing to availability of data, high-performance processors, effective learning algorithms, and easy-to-use programming tools, ANNs have achieved tremendous breakthroughs in many intelligent applications. Recently, SNNs also attracted a lot of attention due to its biological plausibility and the possibility of achieving energy-efficiency (Roy et al., 2019). However, they suffer from ongoing debates and skepticisms due to worse accuracy compared to “standard” ANNs. The performance gap comes from a variety of factors, including learning techniques, benchmarks, programming tools and execution hardware, all of which in SNNs are not as developed as those in the ANN domain.

To this end, we propose a Research Topic, named “Understanding and Bridging the Gap between Neuromorphic Computing and Machine Learning,” in Frontiers in Neuroscience and Frontiers in Computational Neuroscience to collect recent researches on neuromorphic computing and machine learning to help understand and bridge the aforementioned gap. We received 18 submissions in total and accepted 14 of them in the end. The scope of these accepted papers covers learning algorithms, applications, and efficient hardware.

LEARNING ALGORITHMS

How to train SNN models is the key to improve its functionality, thus bridging the gap between ANN models. Unlike the ANN domain that has grown rapidly via sophisticated backpropagation-based learning algorithms, the SNN domain is still short of effective learning algorithms due to the complicated spatiotemporal dynamics and non-differentiable spike activities. Currently, there are overall two categories of learning algorithms for SNNs: unsupervised synaptic plasticity with biological plausibility [e.g., spike timing dependent plasticity, STDP (Diehl and Cook, 2015)] and supervised backpropagation

OPEN ACCESS

Edited and reviewed by:

Si Wu,
Peking University, China

*Correspondence:

Lei Deng
leideng@mail.tsinghua.edu.cn

Received: 08 February 2021

Accepted: 12 February 2021

Published: 17 March 2021

Citation:

Deng L, Tang H and Roy K (2021)
Editorial: Understanding and Bridging
the Gap Between Neuromorphic
Computing and Machine Learning.
Front. Comput. Neurosci. 15:665662.
doi: 10.3389/fncom.2021.665662

with gradient descent [e.g., indirect learning by acquiring gradients from the ANN counterpart (Diehl et al., 2015; Sengupta et al., 2019), direct learning by acquiring gradients from the SNN itself (Lee et al., 2016; Wu et al., 2018; Gu et al., 2019; Zheng et al., 2021), or the combination of both (Rathi et al., 2020)]. The latter can usually achieve higher accuracy and has advanced the model scale to handle ImageNet-level tasks. In the future, we look forward to seeing more studies on SNN learning to close the gap.

Next, we briefly summarize the recent progress of neural network (especially SNN) learning presented in our accepted papers. Inspired by the curiosity-based learning mechanism of the brain, Shi et al. propose curiosity-based SNN (CBSNN) models. In the first training epoch, the novelty estimations of all samples are obtained through bio-plausible synaptic plasticity; next, the samples whose novelty estimations exceed the threshold are repeatedly learned and the novelty estimations are updated in the next five epochs; then, all samples are learned with one more epoch. The last two steps are periodically taken until convergence. CBSNNs show better accuracy and higher efficiency in processing several small-scale datasets than conventional voltage-driven plasticity-centric SNNs. Daram et al. propose ModNet, an efficient dynamic learning system inspired from the neuromodulatory mechanism in the insect brain. An inbuilt modulatory unit regulates learning based on the context and internal state of the system. The network with modulatory trace achieves $98.8\% \pm 1.16$ on average over the omniglot dataset for five-way one-shot image classification task while using 20x fewer trainable parameters compared to state-of-the-art models. Kaiser, Mostafa et al. introduce deep continuous local learning (DECOLLE), an SNN model equipped with local error functions for online learning. The synaptic plasticity rules are derived from user-defined cost functions and neural dynamics by leveraging existing autodifferentiation methods of machine learning frameworks. The model demonstrates state-of-the-art performance on N-MNIST and DvsGesture datasets. Fang et al. propose a bio-plausible noise structure to optimize the performance of SNNs trained by gradient descent. Through deducing the strict saddle condition for synaptic plasticity, they demonstrate that the noise helps the optimization escape from saddle points on high dimensional domains. The accuracy improvement can reach at least 10% on MNIST and CIFAR10 datasets. Panda et al. modify the SNN configuration with backward residual connections, stochastic softmax, and hybrid artificial-and-spiking neuronal activations. In this way, the previous learning methods are improved with comparable accuracy but large efficiency gains over the ANN counterparts.

APPLICATIONS

Unlike the artificial neuron in ANNs, each spiking neuron in SNNs has intrinsic temporal dynamics, which is appropriate for processing sequence information. In this Research Topic, we accepted two papers that discuss SNN applications. Wu et al. explore the first work that uses SNNs for large-vocabulary continuous automatic speech recognition (LVCSR)

tasks. Their SNNs demonstrate competitive accuracies on par with their ANN counterparts while consuming only 10 algorithmic timesteps and $0.68\times$ total synaptic operations. They integrate the models into the PyTorch-Kaldi Speech Recognition Toolkit for rapid development. Kugele et al. apply SNNs for processing spatiotemporal event streams (e.g., N-MNIST, CIFAR10-DVS, N-CARS, and DvsGesture datasets). They improve the ANN-to-SNN conversion learning method by introducing connection delays during the pre-training of ANNs to match the propagation delays in converted SNNs. In this way, the resulting SNNs can handle the above tasks accurately and efficiently.

In addition, besides energy-efficiency (Merolla et al., 2014), recent studies further find that the event-driven computing paradigm of SNNs endows them high robustness (He et al., 2020; Liang et al., 2020) and superior capability in learning sparse features (He et al., 2020). We believe it is very important to mine the true advantages of SNNs to determine their true value in practical applications.

EFFICIENT HARDWARE

Performing neural networks on general-purpose processors is inefficient, which stimulates the development of various domain-specific hardware platforms, including those for ANNs [e.g., DaDianNao (Chen et al., 2014), TPU (Jouppi et al., 2017), Eyeriss (Chen et al., 2017), Thinker (Yin et al., 2017), etc.], for SNNs (e.g., SpiNNaker (Furber et al., 2014), TrueNorth (Merolla et al., 2014), Loihi (Davies et al., 2018), DYNAPs (Moradi et al., 2017)], and for cross-paradigm modeling [e.g., Tianjic (Pei et al., 2019; Deng et al., 2020)]. In this Research Topic, we accepted seven papers for neural network hardware: three for ANNs, two for SNNs, and two for cross-paradigm.

Sim and Lee propose SC-CNN, the bitstream-based convolutional neural network (CNN) inspired by stochastic computing (SC) that uses bitstreams to represent numbers, to improve machine learning hardware. Benefitting from the CNN substrate, SC-CNN can achieve high accuracy; further benefitting from SC, SC-CNN is highly efficient, scalable, and fault-tolerant. Different from the common digital machine learning accelerators, Kaiser, Faria et al. present a clockless autonomous probabilistic circuit, wherein synaptic weights are read out in the form of analog voltages, for fast and efficient learning with no use of digital computing. They demonstrate a circuit built with existing technology to emulate the Boltzmann machine learning algorithm. Muller et al. introduce bias matching, a top-down neural network design approach, to match the inductive biases required in a machine learning system to the hardware constraints of its implementation.

To alleviate the high cost training of SNNs using backpropagation, Lee et al. propose a spike-train level direct feedback alignment (ST-DFA) algorithm. Compared to the state-of-the-art backpropagation learning algorithm, they demonstrate excellent performance vs. overhead tradeoffs on FPGA for speech and image classification applications.

Dutta et al. propose an all ferroelectric field-effect transistors (FeFET)-based SNN hardware that allows low-power spike-based information processing and co-localized memory and computing. They implement a surrogate gradient supervised learning algorithm on their efficient SNN platform, which further accounts for the impacts of device variation and limited bit precision of on-chip synaptic weights on the classification accuracy.

Parsa et al. build a hierarchical pseudo agent-based multi-objective Bayesian hyperparameter optimization framework for both software and hardware. They can not only maximize the performance of the network, but also minimize the energy and area overheads of the corresponding neuromorphic hardware. They validate the proposed framework using both ANN and SNN models, which involves both deep learning accelerators [e.g., PUMA (Ankit et al., 2019)] and neuromorphic hardware [e.g., DANNA2 (Mitchell et al., 2018) and mrDANNA (Chakma et al., 2017)]. Instead of implementing ANNs and SNNs separately, integration of them has become a promising direction to achieve further breakthroughs toward AGI via complementary advantages (Pei et al., 2019). Therefore, the efficient hardware that can support individual modeling of ANNs and SNNs as well as their hybrid modeling is very important. This has been achieved by the cross-paradigm Tianjic platform (Deng et al., 2020), based on which Wang et al. further present an end-to-end mapping framework for implementing various hybrid neural networks. By constructing hardware configuration schemes for four typical signal conversions and establishing a global timing adjustment mechanism among different heterogeneous modules, they implement hybrid models with low execution latency and low power consumption.

REFERENCES

- Ankit, A., Hajj, I. E., Chalamalasetti, S. R., Ndu, G., Foltin, M., Williams, R. S., et al. (2019). "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Providence, RI: ACM), 715–731. doi: 10.1145/3297858.3304049
- Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., and Rose, G. S. (2017). Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level. *IEEE J. Emerg. Selected Topics Circuits Syst.* 8, 125–136. doi: 10.1109/JETCAS.2017.2777181
- Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. (2017). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* 52, 127–138. doi: 10.1109/JSSC.2016.2616357
- Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., et al. (2014). "Dadiannao: a machine-learning supercomputer," in *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Cambridge: IEEE/ACM), 609–622. doi: 10.1109/MICRO.2014.58
- 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
- Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020). Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid-State Circuits* 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709
- 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 *International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696
- 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
- Gu, P., Xiao, R., Pan, G., and Tang, H. (2019). "STCA: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *International Joint Conferences on Artificial Intelligence (IJCAI)*, 1366–1372. doi: 10.24963/ijcai.2019/189
- He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., et al. (2020). Comparing SNNs and RNNs on neuromorphic vision datasets: similarities and differences. *Neural Networks* 132, 108–120. doi: 10.1016/j.neunet.2020.08.001
- 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 (ISCA)* (Toronto, ON: ACM), 1–12. doi: 10.1145/3079856.3080246
- 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
- Liang, L., Hu, X., Deng, L., Wu, Y., Li, G., Ding, Y., et al. (2020). Exploring adversarial attack in spiking neural networks with spike-compatible gradient. *arXiv preprint arXiv:2001.01587*.

CONCLUSION

Machine learning and neuromorphic computing are two modeling paradigms on the road toward AGI. ANNs have achieved tremendous breakthroughs in many intelligent applications benefitting from big data, high-performance processors, effective learning algorithms, and easy-to-use programming tools; in contrast, SNNs are still in its infant stage and there is a dire need for more neuromorphic benchmarks. Through cross-discipline research, we expect to understand and bridge the gap between neuromorphic computing and machine learning. This Research Topic is just a small step in this direction, and we look forward to more innovations that can achieve brain-like intelligence.

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 supported in part by the key scientific technological innovation research project by Ministry of Education of China, Zhejiang Lab (Grant No. 2019KC0AD02), Center for Brain Inspired Computing (C-BRIC), a Semiconductor Research Corporation (SRC) program sponsored by DARPA, National Science Foundation, Sandia National Laboratory, ONR sponsored Multi-University Research Initiative (MURI), and Vannevar Bush Faculty Fellowship program.

- 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
- Mitchell, J. P., Dean, M. E., Bruer, G. R., Plank, J. S., and Rose, G. S. (2018). “DANNA 2: Dynamic adaptive neural network arrays,” in *Proceedings of the International Conference on Neuromorphic Systems (ICONS)* (Knoxville, TN: ACM), 1–6. doi: 10.1145/3229884.3229894
- Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2017). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Trans. Biomed. Circuits Syst.* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700
- 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
- Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). “Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation,” in *International Conference on Learning Representations (ICLR)*.
- 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
- 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, 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
- Yin, S., Ouyang, P., Tang, S., Tu, F., Li, X., Zheng, S., et al. (2017). A high energy efficient reconfigurable hybrid neural network processor for deep learning applications. *IEEE J. Solid-State Circuits* 53, 968–982. doi: 10.1109/JSSC.2017.2778281
- Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. (2021). “Going deeper With directly-trained larger spiking neural networks,” in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press).

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.

Copyright © 2021 Deng, Tang and Roy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



A Curiosity-Based Learning Method for Spiking Neural Networks

Mengting Shi^{1,2†}, Tielin Zhang^{1†} and Yi Zeng^{1,2,3,4*†}

¹ Research Center for Brain-inspired Intelligence, Institute of Automation, Chinese Academy of Sciences, Beijing, China,

² University of Chinese Academy of Sciences, Beijing, China, ³ Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Shanghai, China, ⁴ National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China

Spiking Neural Networks (SNNs) have shown favorable performance recently. Nonetheless, the time-consuming computation on neuron level and complex optimization limit their real-time application. Curiosity has shown great performance in brain learning, which helps biological brains grasp new knowledge efficiently and actively. Inspired by this leaning mechanism, we propose a curiosity-based SNN (CBSNN) model, which contains four main learning processes. Firstly, the network is trained with biologically plausible plasticity principles to get the novelty estimations of all samples in only one epoch; secondly, the CBSNN begins to repeatedly learn the samples whose novelty estimations exceed the novelty threshold and dynamically update the novelty estimations of samples according to the learning results in five epochs; thirdly, in order to avoid the overfitting of the novel samples and forgetting of the learned samples, CBSNN retrains all samples in one epoch; finally, step two and step three are periodically taken until network convergence. Compared with the state-of-the-art Voltage-driven Plasticity-centric SNN (VPSNN) under standard architecture, our model achieves a higher accuracy of 98.55% with only 54.95% of its computation cost on the MNIST hand-written digit recognition dataset. Similar conclusion can also be found out in other datasets, i.e., Iris, NETtalk, Fashion-MNIST, and CIFAR-10, respectively. More experiments and analysis further prove that such curiosity-based learning theory is helpful in improving the efficiency of SNNs. As far as we know, this is the first practical combination of the curiosity mechanism and SNN, and these improvements will make the realistic application of SNNs possible on more specific tasks within the von Neumann framework.

OPEN ACCESS

Edited by:

Huajin Tang,
Zhejiang University, China

Reviewed by:

Radwa Khallil,
Jacobs University Bremen, Germany
Malu Zhang,
National University of Singapore,
Singapore

*Correspondence:

Yi Zeng
yi.zeng@ia.ac.cn

[†]These authors have contributed
equally to this work

Received: 10 June 2019

Accepted: 20 January 2020

Published: 07 February 2020

Citation:

Shi M, Zhang T and Zeng Y (2020) A
Curiosity-Based Learning Method for
Spiking Neural Networks.
Front. Comput. Neurosci. 14:7.
doi: 10.3389/fncom.2020.00007

Keywords: curiosity, spiking neural network, novelty, STDP, voltage-driven plasticity-centric SNN

1. INTRODUCTION

As neural networks are inspired by the brain at multiple levels and show higher accuracy and wider adaptability compared with algorithms with fixed parameters, they have become one of the important methods for the development of artificial intelligence. The deep neural network (DNN) inspired by the visual cortex has demonstrated its effectiveness in many aspects, such as: visual tasks (He et al., 2017), audio recognition (Audhkhasi et al., 2017), natural language processing (Yogatama et al., 2018), reinforcement learning (Pathak et al., 2017) and etc. However, due to the poor adaptability and interpretability of traditional Artificial Neural Networks (ANNs), more studies have focused on Spiking Neural Networks (SNNs) whose computational units (e.g., Leaky Integrate and Fire Model Gerstner and Kistler, 2002, Hodgkin-Huxley Model, Izhikevich Model Izhikevich, 2003, and Spike Response Model Gerstner, 2001) and plastic learning methods

(e.g., Spike-Timing-Dependent Plasticity Dan and Poo, 2004; Frémaux and Gerstner, 2016 and Hebbian learning Song et al., 2000) are more similar to that of the human brain, making it more potential to achieve high levels of cognitive tasks (Maass, 1997; Zenke et al., 2015; Khalil et al., 2017b).

At present, SNNs have been well implemented in some brain regions modeling and cognitive functions simulation, like image classification (Zhang et al., 2018b), working memory maintenance (Zhang et al., 2016), decision-making tasks (Héricé et al., 2016; Zhao et al., 2018), cortical development (Khalil et al., 2017a), contingency perception (Pitti et al., 2009) etc. However, even though the training methods proposed by Zhang et al. (2018a) and Shrestha and Orchard (2018) make SNNs performance comparable to ANNs, they are at the cost of a large amount of time. This is because: (1) the network has a certain degree of overfitting problem when fed with a large number of training samples passively; (2) the SNN's training itself is difficult which needs to process sequential spiking signals; (3) until now, the SNNs are still running on the von Neumann framework instead of the specific designed neural chips, which makes the simulation of neurons inefficient, since the information transformation between CPU and memory usually cost too much time.

Traditional learning methods typically get representations of training data from stationary batches, with little regard to the fact that information becomes incrementally available over time (Parisi et al., 2019). A system with brain-inspired intelligence should be composed of inputs, outputs, and plastic components that change in response to experiences in an environment, and autonomously discover novel adaptive algorithms (Soltoggio et al., 2018).

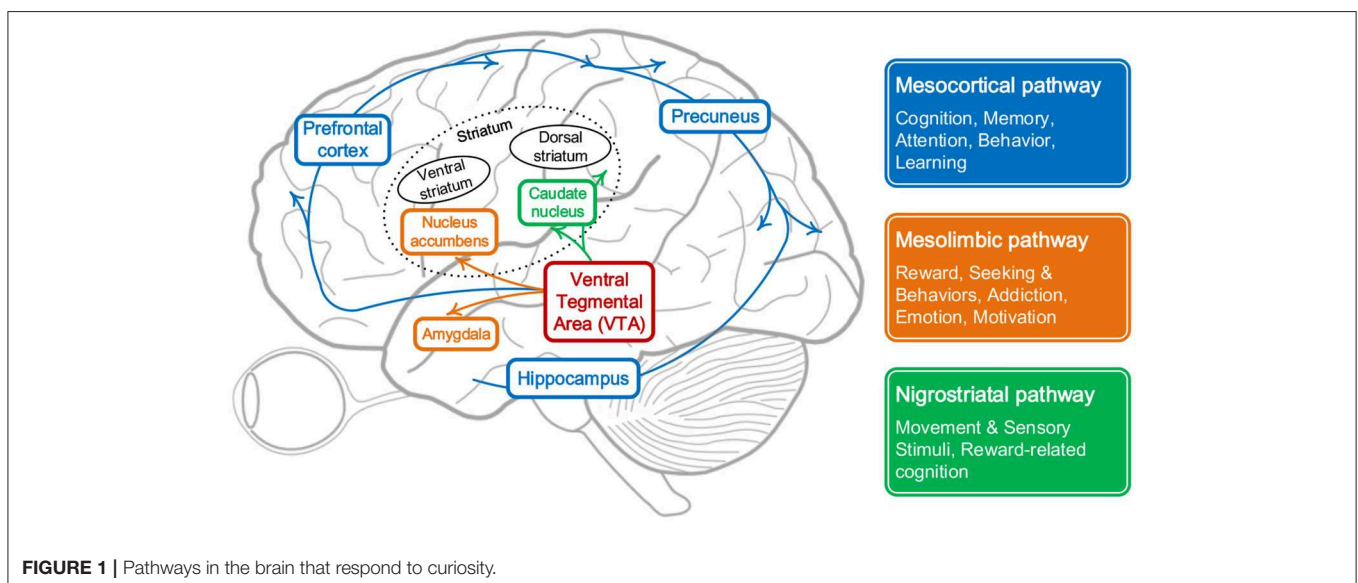
While the curiosity-based learning system in the brain helps us to grasp new knowledge efficiently and actively. From the microscopic point of view, as shown in **Figure 1**,

the continuity of cognitive process in the brain sometimes may be interrupted by some specific unfamiliar or uncertain stimulus, which are mostly from the response of mesolimbic pathway. This pathway is reward pathway (Dreyer, 2010), which connects the ventral tegmental area in the midbrain, to the ventral striatum of the basal ganglia in the forebrain (Ikemoto, 2010) and starts to release neurotransmitters when facing unfamiliar information, like dopamine, serotonin, and opioid which could regulate characteristics associated with curiosity, like:

Memory: The novelty of stimuli can be considered as the result of continual comparison between the current state and previous experiences, which will cover the brain regions related to long-term and short-term memory, e.g., the hippocampus and parahippocampus gyrus. After the comparison, individuals can give a corresponding level of novelty for specific stimuli (Sahay et al., 2011).

Attention and Learning: With the limitation of energy and efficiency of the biological system, attention plays a vital role in focusing on the stimuli most important or relevant. Some patients with a degenerative disease, for example, Alzheimer's disease, show bad performance on identifying novel stimuli, during which cells in some brain regions, like hippocampus, don't run well and thus prevent the communication with assessing or rewarding process. Attention is a continuous and gradual learning process during which striatum and precuneus get involved in influencing levels of curiosity in terms of novelty (Zola et al., 2011).

Motivation: Curiosity has been described as a desire for learning and knowledge, especially what is unknown (Kang et al., 2009). The idea that dopamine modulates novelty seeking is supported by evidence that novel stimuli excite dopamine neurons and activate brain regions receiving dopaminergic input. In an fMRI study, activation in ventral striatum encoded both standard reward prediction errors and enhanced reward



prediction errors during novelty driven exploration (Costa et al., 2014).

Curiosity-based exploration behaviors depend on the estimation of difficulty or novelty of tasks, which are related to one previous learning situation and would update gradually (Faraji et al., 2018). An intuitive paradigm that may enlighten us is designed in Baranes et al. (2014), during which the subjects are allowed to choose games with different levels to get high score freely. After mastering simple skills, people are more likely to repeat hard or novel games frequently instead of spending time on overly comfortable or familiar experiments. This result shows that the difficulty and novelty of tasks have a significant impact on the motivation of exploration.

In this paper, we propose a curiosity-based SNN (CBSNN), and the learning process of this model includes four steps:

- Step (1): Before the predefined starting time, the CBSNN is trained with a traditional method to get the novelty estimations of all samples in only one epoch;
- Step (2): Once the current iteration time is over the starting time, the CBSNN begins to repeatedly learn the samples whose novelty estimations exceed the novelty threshold and dynamically update the novelty estimations of samples according to the learning results within the retrain interval (we use five epochs later);
- Step (3): When the duration of step (2) reaches the retrain interval, the CBSNN retrains all samples once (one epoch) in order to avoid the overfitting of the novel samples and forgetting of the learned samples.
- Step (4): The model repeats step (2) and (3) until the algorithm converges.

The MNIST hand-written digit recognition dataset is used to verify the performance of our proposed model. Through a series of experiments, we analyze how the proposed method affects the computation efficiency and learning accuracy of the traditional SNN. By comparing with the state-of-the-art Voltage-driven Plasticity-centric SNN (VPSNN) (Zhang et al., 2018a) under standard architecture, our model achieves higher accuracy of 98.55% with only 54.95% of its computation resources.

2. RELATED WORKS

Several curiosity-related works have been proposed in different research areas, which include but are not limited to active learning, curriculum learning, sample selection strategies, and reinforcement learning.

Active learning is good at select discriminating samples dynamically from large training data sets and training the model efficiently (Zhou et al., 2017). It pays more attention to some informative and representative data to overcome the labeling bottleneck (Konyushkova et al., 2017). However, the curiosity-based learning strategy dynamically evaluates the difficulty or novelty of the sample and makes a selection based on the current learning situation of the network. The quality of learning results

not only depends on the representativeness of the sample itself, but also is more related to the specific learning process.

Bengio et al. (2009) proposed curriculum learning to imitate the characteristics of human learning process, and let the model learn from simple to difficult in multiple stages (Ugur et al., 2007; Chernova and Veloso, 2009). It defines the difficulty level of sample before training, and gives the initial weight distribution. However, in curiosity-based learning process, we tend to predefine an evaluation function, which could be many forms, and let the model adjust dynamically.

Cheng et al. (2018) proposed an active sample selection strategy that reaches state-of-the-art accuracy on visual models ResNet-50, DenseNet-121, and MobileNet-V1, which has a lower computation cost compared with previous networks.

Besides, Schmidhuber (1991a,b) used adaptive “world model” to implement neural controllers and reinforcement learning. The system is “curious” in the sense that it described how the particular algorithm may be augmented by dynamic curiosity and boredom in a natural manner. Pathak et al. (2017) introduced a curiosity assessment module which represents the difference between predicted situation and real situation as an intrinsic reward signal to make agents complete games quicker than just using external reward. Savinov et al. (2019) further uses this strategy to pay more attention to those remarkable situations in order to speed up the completion of tasks by agents and avoid the model falling into local optimum to a certain extent. Inspired by the infants’ ability to generate novel structured behaviors in unstructured environments that lack clear extrinsic reward signals, Haber et al. (2018) mathematically modeled this mechanism using a neural network that implements curiosity-driven intrinsic motivation to create a self-supervised agent.

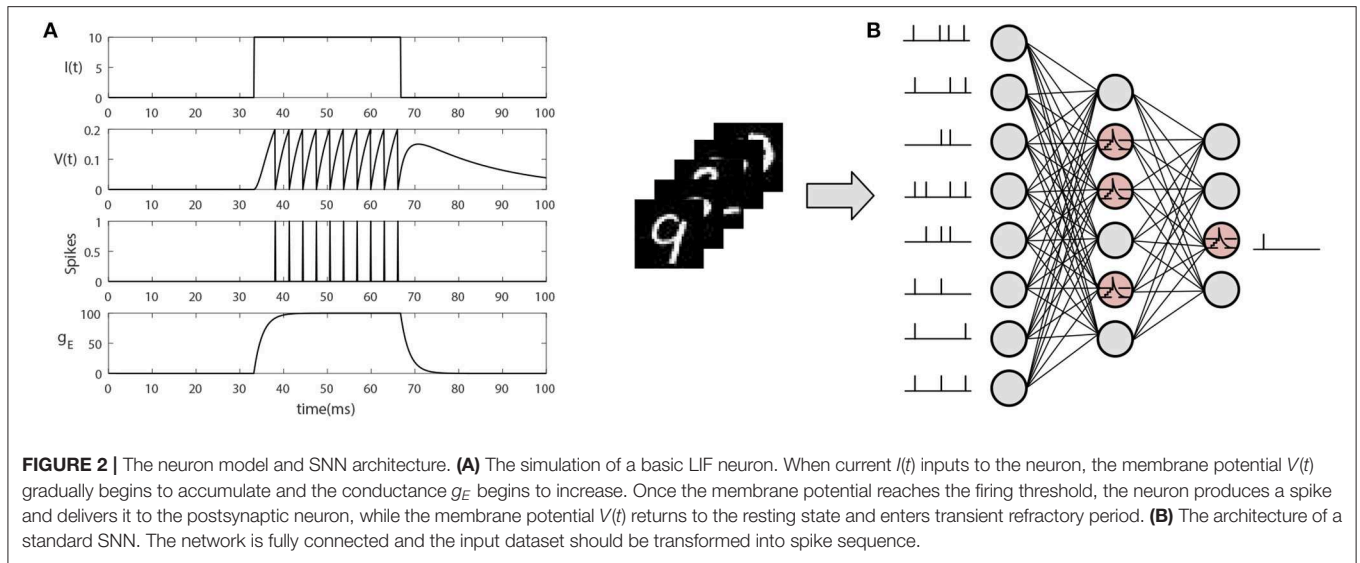
However, most of current studies only discuss the possibility of application and performance improvement of curiosity-based learning mechanism under the traditional ANNs’ framework. In this paper, we try to combine this brain-inspired curiosity-based learning mechanism with more biologically plausible SNN to improve its current problems in computation efficiency under the traditional computing system, so that it can be applied more widely in the future.

3. METHODS

In this section, we will introduce the network architecture (including neuron model and network structure) and the learning process of CBSNN in detail.

3.1. The Architecture of CBSNN

The network should be designed with different neuron model, synapse model, network structure or learning method in order to solve different tasks. Diehl and Cook (2015) designed a simple two-layered SNN to achieve MNIST (LeCun, 1998) classification. Zhang et al. (2016) had a recurrent part to store memory and eliminate noise. To have a better performance on



complex dataset, Shrestha and Orchard (2018) used feed-forward and back propagation procedure at the same time.

In this paper, we adopt a standard three-layered structure, which is similar to Zhang et al. (2018a) to verify the validity of the curiosity-based mechanism in training SNN.

3.1.1. The Neuron Model

Here we adopt the Leaky integrate-and-fire (LIF) neuron model as the basic processing unit. In the LIF model, the neuron will be regarded as a node. Regardless of the transmission of electrical signals in neurons, the variation of the potential difference $u(t)$ between inside and outside the membrane at time t satisfies the Equation (1).

$$C_m \frac{du(t)}{dt} = -\frac{u(t)}{R_m} + I(t) \quad (1)$$

where C_m is the membrane capacitance in which m is the abbreviation of membrane, R_m is the membrane resistance, and $I(t)$ is the weighted sum of all input currents (the weight is usually the connection value $w_{i,j}$ between neuron i and j). If we use $V(t)$ to denote membrane potential, V_L to denote leaky potential, g_L to denote leaky conductance, then we could have Equation (2) which demonstrates the change of membrane potential.

$$C_m \frac{dV(t)}{dt} = -g_L(V(t) - V_L) + I(t) \quad (2)$$

Under the consideration of real brain, we introduce excitatory conductance g_E and excitatory reversal potential V_E and we can have the membrane potential updating Equation (3) based on excitatory conductance.

$$\begin{cases} \tau_E \frac{dg_E}{dt} = -g_E + \eta \sum_{j \in N_E} w_{j,i} \delta_t \\ \tau_m \frac{dV(t)}{dt} = -(V(t) - V_L) - \frac{g_E}{g_L}(V(t) - V_E) \end{cases} \quad (3)$$

When membrane potential integrates up to the firing threshold V_{th} , the neuron produces a spike, and sends it to postsynaptic

neurons. After that, the membrane potential is reset to resting state and the neuron enters into refractory time. The simulation results are shown in Figure 2A.

3.1.2. The Network Structure

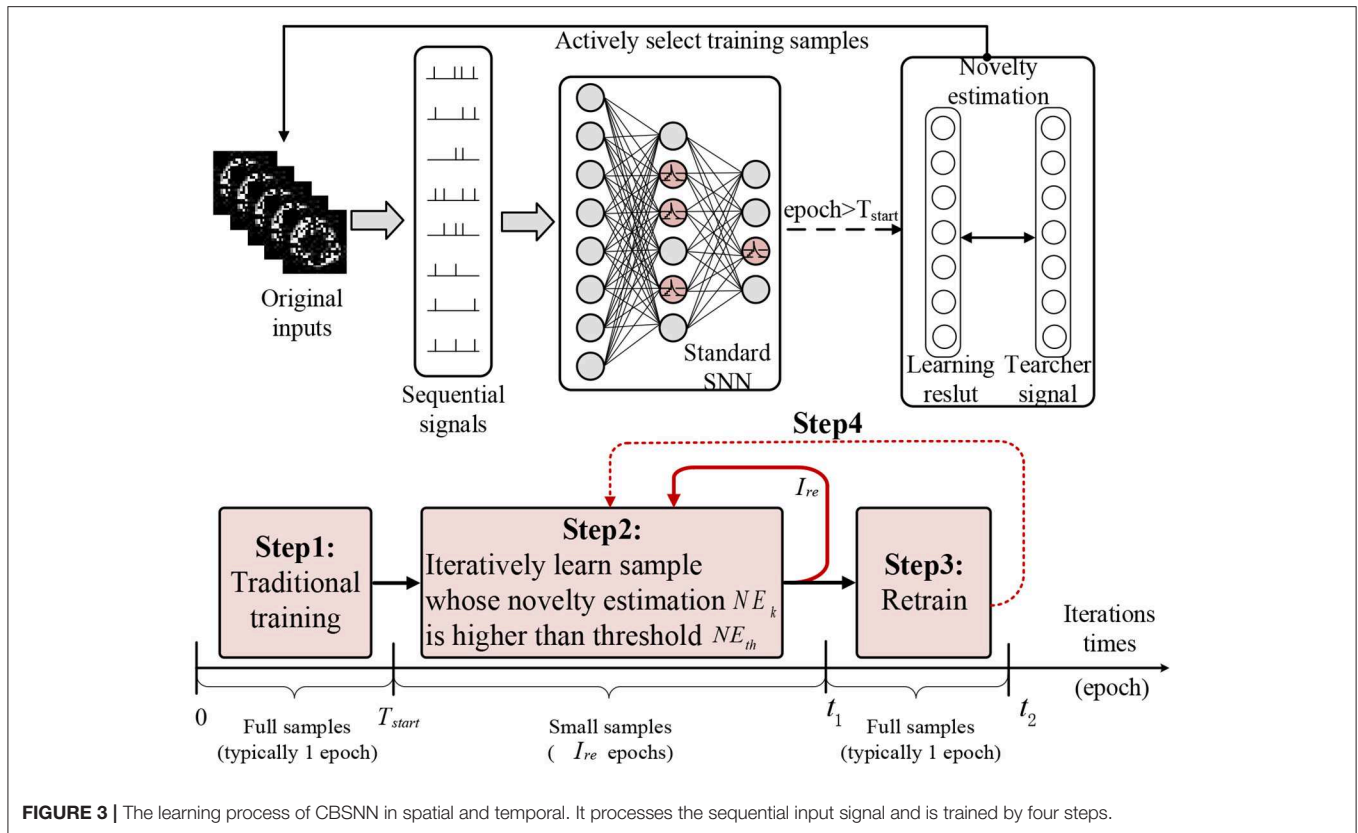
In this paper, we adopt a standard three layers SNN like (Zhang et al., 2018a). As shown in Figure 2B, the first layer receives sequential signals converted from original dataset; the second layer abstracts the input information using non-linear characteristic; the third layer produces the final classification signals which will be used to transmit error signals and generate novelty estimates with the help of the teacher signals.

3.2. The Learning Method of CBSNN

In this section, we will introduce the detailed curiosity-based learning method on SNN (CBSNN). As shown in Figure 3, we first transform the original input data into sequential signals. Then learning process of CBSNN contains four steps: (1) Before the starting time T_{start} of sample selection (we use one epoch later), the CBSNN is able to train all examples in order to get the novelty estimation of whole dataset; (2) Once the current iteration time is over the predefined starting time T_{start} , the CBSNN begins to repeatedly select the sample whose novelty estimation NE_k exceeds the threshold NE_{th} , and dynamically update the novelty estimations NE_k of samples according to the learning results within the retrain interval I_{re} (we use 5 epochs later); (3) When the duration of step (2) reaches the retrain interval I_{re} , the CBSNN retrains all data once (one epoch) in order to avoid the overfitting of the novel samples and forgetting of the learned samples; (4) the model repeats step (2) and (3) until the algorithm converges.

3.2.1. Step 1: Traditional Training Before Starting Time T_{start}

At the beginning of learning, we put all data into the SNN before the starting time T_{start} . With sequential signals passing in feed forward, the change of membrane potential ΔV_i^{FF} of neuron i



and excitatory conductance Δg_E in this stage are first updated by Equation (3). Then according to the equilibrium tuning which is one of the efficient ways to solve the non-differential problem of SNN (Scellier and Bengio, 2017), the membrane potential is changed again by Equation (4) (Zhang et al., 2018a).

$$\Delta V_i^{ES} = -\eta_i \frac{V_i - (\sum_j^N w_{j,i} V_j - \sum_j^N V_{th,i})}{-(V_i - V_L) - \frac{g_E}{g_L}(V_i - V_E)} \quad (4)$$

Combining the result of these two stages shown in Equation (5), we get the final change of membrane potential of neuron i in an unsupervised way.

$$\Delta V_i = \frac{t}{T} \Delta V_i^{FF} + (1 - \frac{t}{T}) \Delta V_i^{ES} \quad (5)$$

In order to let the model have a better performance and calculate the novelty estimation of samples, we introduce the teacher signal V_T . By optimizing the loss function $C = \sum_{i=1}^{L_3} (V_i - V_T)^2$, the membrane potentials of final layer are changed as Equation (6) in supervised way.

$$dV_i = -\eta^c (V_i - V_T) \quad (6)$$

3.2.2. Step 2: Curiosity Learning Based on Novelty Estimation NE and Novelty Threshold NE_{th}

The weights among these three layers can be updated with multiform Spike-Timing-Dependent Plasticity (STDP) (Dan and

Poo, 2004). Here we use a simple but effective one: bi-STDP. Once the current iteration is up to the predefined starting time T_{start} , the weights of the model will be passively changed through Equation (7) (V'_i is the derivation of V_i).

$$\Delta w_{j,i} \propto V_j V'_i \quad (7)$$

After the updating of weights, we should get the assessment of samples' learning situation in order to provide an efficient way to select appropriate samples to train in the next iterations. According to the curiosity theory, humans tend to explore novel and difficult problems rather than spend time on general and simple samples in the learning process. Inspired by this, we define the novelty estimation NE for samples during the SNN learning process. As shown in Equation (8), instead of error rate, we adopt a similarity evaluation method: the cosine distance between training outputs V_k and the corresponding teacher signals V_T , to get more concrete novelty estimation of the sample k .

$$NE_k(V_k, V_T) = 1 - \cos \langle V_k, V_T \rangle = 1 - \frac{V_k \cdot V_T}{\|V_k\| \|V_T\|} \quad (8)$$

According to the novelty estimation NE_k of the sample k and predefined novelty threshold NE_{th} , we can obtain the sample selection strategy as shown in Equation (9). And when $S(k)$ equals to 1, the k th sample is selected. Then, the CBSNN repeatedly trains the samples whose novelty estimations exceed

the threshold NE_{th} and temporarily ignores the simple samples which have already learned well, and dynamically updates the novelty estimations of samples using Equation (8) based on the learning results.

$$S(k) = \begin{cases} 1 & NE_k \geq NE_{th} \\ 0 & NE_k < NE_{th} \end{cases} \quad (9)$$

3.2.3. Step 3: Anti Overfitting and Catastrophic Forgetting Based on Retrain Interval I_{re}

If the model only selects novel samples to train in every iteration, it is inevitable to cause overfitting of novel samples and forgetting of simple samples. So the model needs to review the whole dataset (novel and non-novel) once the duration of dynamic training of step (2) reaches the retrain interval I_{re} (Kirkpatrick et al., 2017). The more detailed learning process of CBSNN is shown in Algorithm 1.

Algorithm 1 The learning process of CBSNN

1. Initialize $T_{start} = 1$, $NE_{th} = 0.05$, $I_{re} = 5$ and other parameters of the network.
2. Load dataset (X, Y)
3. Start training procedure

$X_s \leftarrow X, Y_s \leftarrow Y, e_0 \leftarrow T_{start}$
for every epoch e **do**

if $e \leq T_{start}$ **then**.....step 1 ∇

SNNTraining(X_s, Y_s , fullsample=True)

end if

if $e > T_{start}$ **then**

if $e - e_0 \neq I_{re}$ **then**.....step 2 ∇

$NE(V_k, V_T) \leftarrow 1 - \cos < V_k, V_T >$ Equation (8)
 $(X_s, Y_s) \leftarrow \text{select}(NE(V_k, V_T) \geq NE_{th})$
 Equation (9)

SNNTraining(X_s, Y_s , fullsample=False)

else.....step 3 ∇

$X_s \leftarrow X, Y_s \leftarrow Y, e_0 \leftarrow e$
 SNNTraining(X_s, Y_s , fullsample=True)

end if
end if

Function: SNNTraining(X_s, Y_s , fullsample=True)

- for** every batch b **do**

for every differential time Δt **do**

$\Delta V_{i,b}^{FF} \leftarrow \text{Feed forward } (X_{s,b})$ by Equation (3)

$\Delta V_{i,b}^{ES} \leftarrow \text{Equilibrium state } (V_{i,b})$ by Equation (4)

$\Delta V_{i,b} \leftarrow \frac{t}{T} \Delta V_{i,b}^{FF} + (1 - \frac{t}{T}) \Delta V_{i,b}^{ES}$ Equation (5)

$\Delta V'_{i,b} \leftarrow \text{supervised tuning } (V_{i,b})$ by Equation (6)

end for
- end for**
- Passively update weights by Equation (7)
-

4. EXPERIMENTS

In this section, we verify the effectiveness of CBSNN and analyze how the proposed method affects the computation efficiency and learning accuracy of the traditional SNN. And all of our experimental results are based on MNIST.

4.1. Hyperparameter Configuration on MNIST

It is hard to get the best values with an exhaustive search for the limitation of computation cost, especially when given a large network. Here we firstly get the best hyperparameters from a smaller network and then apply them to a larger network. The hyperparameters includes NE_{th} , retrain interval I_{re} and T_{start} . We set an initial CBSNN which is VPSNN with 200 hidden neurons, novelty threshold $NE_{th} = 0.05$, retrain interval $I_{re} = 5$, starting time $T_{start} = 1$. And the following analysis only changes the corresponding parameters on the basis of this initial CBSNN. The specific computation efficiency is calculated by the ratio of the computational time cost of CBSNN and VPSNN.

4.1.1. Starting Time T_{start}

Before the starting time T_{start} , the network is trained to get the novelty estimate of all samples based on the teacher's signal in few epochs. After that, the network starts to select samples through the novelty threshold and dynamically updates their novelty estimation in every iteration. From **Figure 4A**, we can see that the starting time has little effect on the accuracy which is basically stable at about 0.98. While, the computation in **Figure 4B** has significant proportional increase when starting time changes from 5 to 50. That means the starting time is robust to the accuracy and can greatly improve the computation efficiency with small value. The result reveals that there is no use to pour the whole data set into the network during all iterations. The earlier the sampling based on novelty estimation, the higher the computation efficiency of the model. And this is the main motivation that we set a small starting time (one epoch) as the hyperparameter in step one of the curiosity-based learning process.

4.1.2. Novelty Threshold NE_{th}

In CBSNN, the novelty threshold NE_{th} determines the volume of the difficult samples which will be repeatedly learned. The definition of novelty threshold depends on the difficulty and scale of tasks. In step 1 and step 3, the NE_{th} is 0 because we need to learn the features of all samples, and in step 2, we set the NE_{th} changing from 0.01 to 0.25 for getting the best threshold which could balance the learning accuracy and computation cost. As shown in **Figure 5**, the larger novelty threshold, the more computation ratio and the lower accuracy we will get. The reason is that the large novelty threshold causes the network to repeatedly learn many simple samples, which leads to overfitting of simple samples and wastes a large amount of computation. Especially, the performance of CBSNN is better

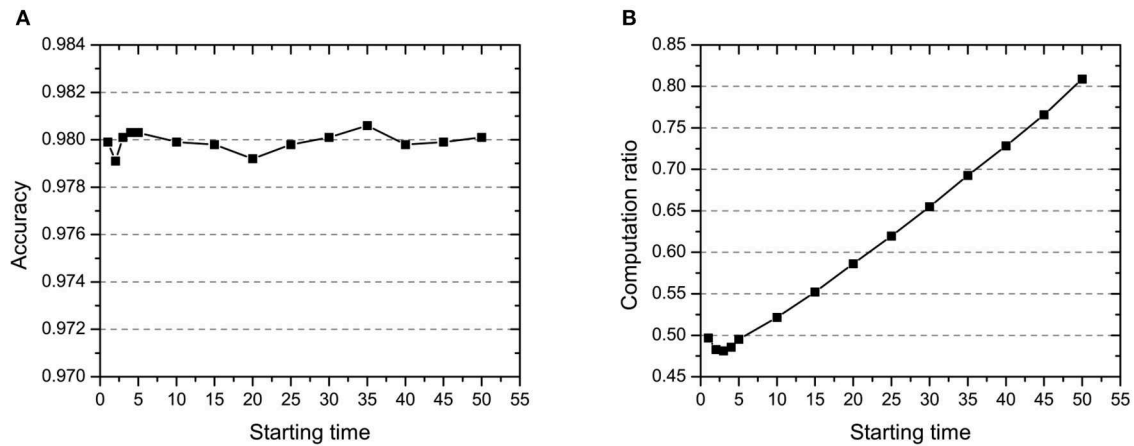


FIGURE 4 | The effect of starting time T_{start} on (A) accuracy and (B) computation ratio.

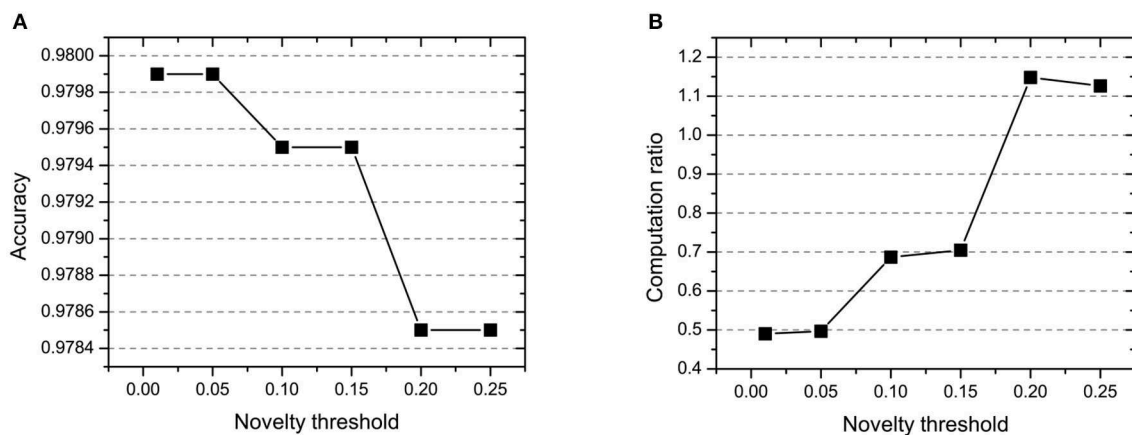


FIGURE 5 | The effect of novelty threshold NE_{th} on (A) accuracy and (B) computation ratio.

than VPSNN in all conditions which shows the effectiveness of the novelty threshold.

4.1.3. Retrain Interval I_{re}

The retrain interval affects the frequency of retraining of all samples. As shown in **Figure 6**, the retrain interval changes from five epochs to 50 epochs and is inversely proportional to accuracy and computation ratio. This parameter leads to a significant decrease in computation (30%), while the accuracy has a little decrease (1%). Especially, even if the retrain interval equals to 50 epochs (only retrain all samples 2–3 times during the whole training), the model can still reach 0.9708 accuracy with 23.75% computation.

To sum up, all parameters have a significant contribution to improving accuracy and reducing computation ratio. And the combination of these parameters is a complex nonlinear relationship. When the CBSNN has comparable accuracy with the VPSNN, the increase in the starting time and the novelty threshold results in a rise in the amount of computation,

and the increase in the retrain interval brings about a computation saving.

4.2. Performance of CBSNN on MNIST

The CBSNN has three main parameters: starting time T_{start} , novelty threshold NE_{th} and retrain interval I_{re} . Combining with the above parameter analysis, we finally obtain a set of parameters with high accuracy and low computational complexity that is starting time $T_{start} = 1$ epoch, novelty threshold $NE_{th} = 0.05$, and retrain interval $I_{re} = 5$ epochs.

Based on the optimal combination of parameters, we compare several different strategies to outstand the effectiveness of our approach further. **Table 1** shows the comparisons of accuracy and corresponding computation ratio based on the different strategies. The comparisons from the first row to fourth row is based on the 200 hidden neurons. Our best result is 0.16% higher than the original VPSNN, and only requires 49.68% computation. When original VPSNN trained with all data costs around 49% computation, the accuracy of it decreases to

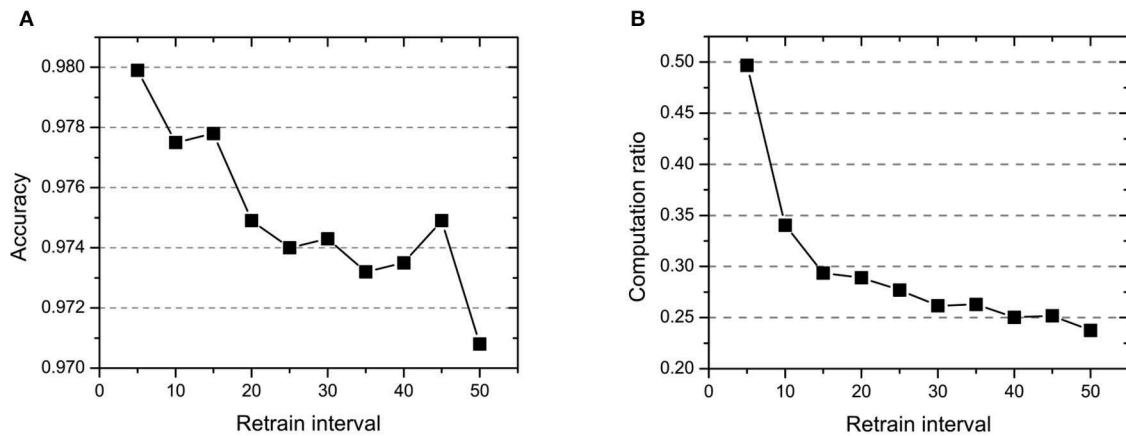


FIGURE 6 | The effect of retrain interval I_{re} on (A) accuracy and (B) computation ratio.

TABLE 1 | The comparison of different strategies.

Strategy	Hidden neurons	Accuracy	Computation ratio (%)
VPSNN	200	0.9783	100
VPSNN with 49% computation	200	0.9773	49.74
VPSNN with 49% random data	200	0.9730	49.43
CBSNN (ours)	200	0.9799	49.68
VPSNN (best) (Zhang et al., 2018a)	4500	0.9852	100
CBSNN (our best)	3000	0.9855	54.95

First part is the comparison of classification accuracy and computation ratio among different sample selecting strategies under 200 hidden neurons. The computation ratio baseline is the VPSNN trained with whole MNIST dataset. The second part is the comparison of the best results of VPSNN and CBSNN. The results of our proposed method in this paper are highlighted in bold.

0.9773. When VPSNN is trained by 49% random data, there is a drop of 0.69% in accuracy compared with CBSNN which means curiosity-based learning method is important to actively dig difficult samples. The fifth row of **Table 1** shows the best accuracy of VPSNN (0.9852) with 4,500 hidden neurons while our proposed CBSNN achieves 0.9855 accuracy with only 54.95% of VPSNN computation.

In order to compare with VPSNN in large-scaled architecture, we set the hidden neurons from 100 to 5,000 and keep starting time $T_{start} = 1$ epoch, novelty threshold $NE_{th} = 0.05$ and retrain interval $I_{re} = 5$ epochs. As shown in **Figure 7**, CBSNN can basically reproduce VPSNN accuracy at every level of the number of hidden neurons, and extremely save the computation (at least 25%). The best accuracy of CBSNN is 0.9855 with 3,000 neurons which is better than VPSNN.

4.3. Analysis of the Inner State of CBSNN on MNIST

The hidden layer and output layer represent highly abstract features which could account for the specific learning situation. t-SNE (Maaten and Hinton, 2008) can decrease high-dimensional data into two or three dimensions and maintain the relationship

of original data as much as possible. Here we use it to observe the change of relationship among all samples when passing these two layers and analyze why our proposed method works during the learning process.

As shown in **Table 2**, every point represents a sample and different colors represent different classes. We set two different sets of parameters for CBSNN. In first group, we have 400 hidden neurons, starting time $T_{start} = 1$ epoch, novelty threshold $NE_{th} = 0.05$ and retrain interval $I_{re} = 5$ epochs. After the first step of CBSNN, the computation ratio is 0.92%, the accuracy rate is 0.9540 and we get the initial novelty estimation of all samples. At this time, we can see that most of the samples have already formed different clusters but some of them are still very discrete and could not be well classified. Then the CBSNN begins to dig out the difficult samples. During this learning process, those discrete and difficult samples are gradually being better learned. Finally, all samples can be well classified and our model reaches 0.9836 accuracy while computation is 40.43%. At this time, the distance among different clusters is larger, and the distance among samples in each cluster is closer. While the original VPSNN with 400 hidden neurons only has 0.9826 accuracy. In second group, we have 400 hidden neurons, starting time $T_{start} = 1$ epoch, novelty threshold $NE_{th} = 0.25$ and retrain interval $I_{re} = 50$ epochs. Under this configuration, the CBSNN could learn faster but have lower accuracy. Compared with the first group, there are more points being misclassified in every iteration time, and the distance between different classes is closer. When the iteration time is 150, the second group only accounts for 20.31% of the computation ratio, but has 0.9753 accuracy which is lower than that of the first group when it reaches 14.97% computation ratio. Experiments show that the optimized and balanced parameter combination can improve the learning rate and accuracy, and also demonstrate the effectiveness of CBSNN.

4.4. The Validation of CBSNN on Other Datasets

In this section, we will discuss how CBSNN performs in more applications. Here we adopted Iris, NETtalk, Fashion-MNIST and CIFAR-10 datasets and in each task, CBSNN and VPSNN

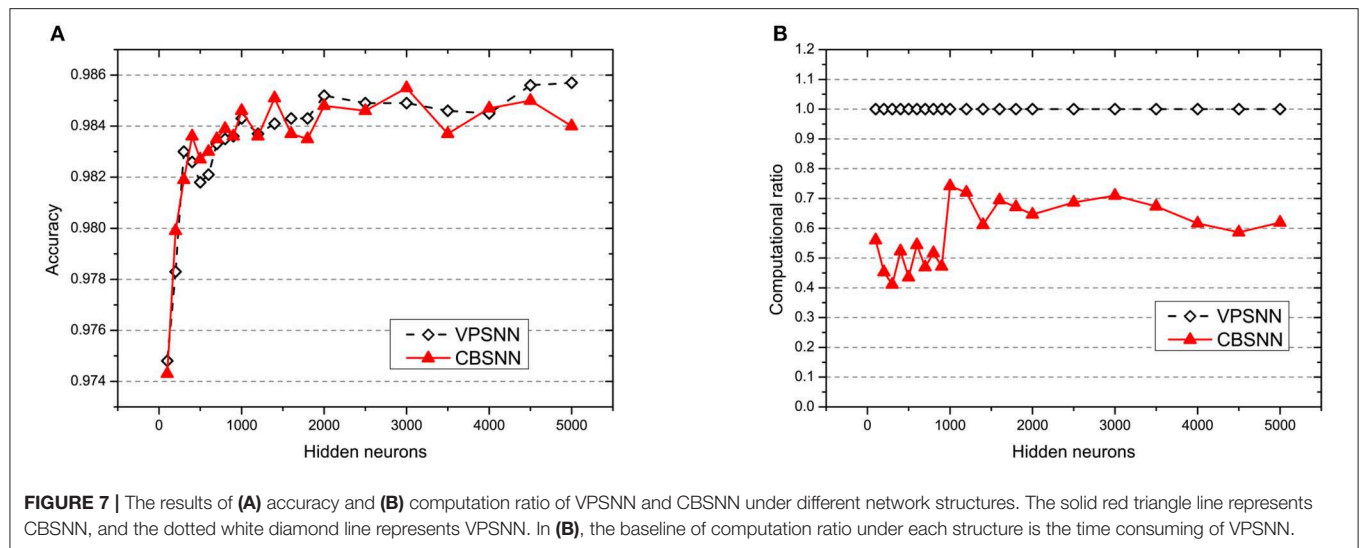
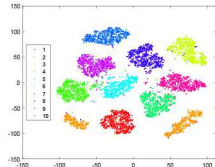
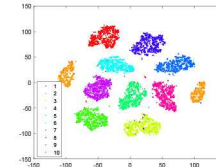
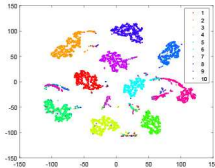
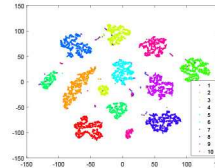
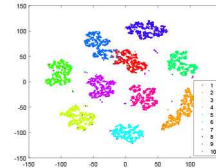
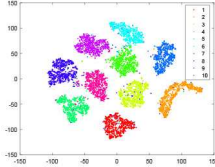
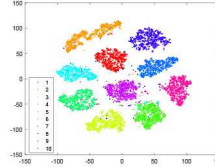
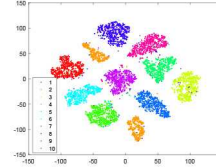
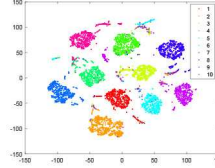


TABLE 2 | The t-SNE visualization of CBSNN learning process with two sets of different parameters.

Iteration time	1	50	100	150
Parameter setting one: hidden neurons = 400, $T_{start} = 1$, $NE_{th} = 0.05$, $I_{re} = 5$				
Hidden layer				
Output layer				
Computation ratio	0.92%	14.97%	27.88%	40.34%
Accuracy of CBSNN	0.9540	0.9813	0.9830	0.9836
Parameter setting two: hidden neurons = 400, $T_{start} = 1$, $NE_{th} = 0.25$, $I_{re} = 50$				
Hidden layer				
Output layer				
Computation ratio	1.24%	9.99%	15.12%	20.31%
Accuracy of CBSNN	0.9614	0.9708	0.9730	0.9753

It shows the results of hidden layer, output layer, computation ratio (the baseline is VPSNN with 0.9826 accuracy and 100% computation cost) and accuracy of CBSNN after different iteration times.

TABLE 3 | The comparison of VPSNN and CBSNN on different tasks.

Dataset	Preprocessing	Network architecture (input-hidden-output)	CBSNN Accuracy	VPSNN Accuracy	Computation ratio (based on VPSNN)
Iris	None	4-2-3	1.0000	0.9667	64.59%
NETtalk	None	189-80-26	0.8720	0.8680	48.08%
Fashion-MNIST	None	784-400-10	0.8574	0.8299	18.35%
	Gray	1024-500-10	0.3198	0.2947	71.69%
CIFAR-10	PCA	700-400-10	0.2546	0.2384	15.51%
	Conv	300-100-10	0.5285	0.5134	31.11%

In each task, CBSNN and VPSNN have the same network architecture and the accuracy of CBSNN (highlighted in Bold) is higher than that of VPSNN with lower computational cost (listed in the last column).

share the same network architecture. The corresponding results are shown in **Table 3**.

- Iris (Fisher, 1936) is a machine learning dataset for multiple variable analysis and contains 120 samples of three classes of Iris flower. We randomly separated it into 90 for training and 30 for test. Finally, CBSNN performs 100% classification accuracy with lower computation cost than VPSNN.
- NETtalk (Sejnowski and Rosenberg, 1987) is usually used for speech generation, consisting 5,033 training and 500 test. The input is a string of letters with fixed length of 7, which is encoded into 189 dimensions (each character has a 27 length one-hot vector). The output is 26 dimensions which represent 72 phonetic principles. For this mapping task with strong global regularities, VPSNN reaches 0.8680 accuracy. Although CBSNN is only slightly higher than VPSNN, it saves about half of the computation cost.
- Fashion-MNIST (Xiao et al., 2017) is more discrete and includes more semantic information than MNIST. It consists of 28*28 gray-scale images of 10 categories of objects in wearing, divided into 60,000 training samples and 10,000 test samples. From **Table 3**, CBSNN reaches the accuracy of 85.74% (higher than VPSNN 2.75%) with only 18.35% computation cost on it.
- CIFAR-10 (Krizhevsky and Hinton, 2009) contains 60,000 samples (50,000 for training and 10,000 for test) and has image size of 32*32 pixels with three channels, which will bring the growth of calculation exponentially and exceed the ability of these two networks. We used some dimension reduction methods for preprocessing it, i.e., RGBtoGray, Principal Components Analysis (PCA) and Convolution. From **Table 3**, the Convolution which converts the original data from 32*32*3 into 300 dimensions, works best and helps CBSNN achieve the best accuracy of 52.85% under only around a third of computation cost of VPSNN.

5. DISCUSSION

SNN is the third-generation neural network (Maass, 1997). It has more biological structures and processing mechanisms, such as discrete sequential spike neurons which make it possible to deal with spatiotemporal information simultaneously, and non-BP biological plasticity like STDP. Although both SNN and ANN are black boxes at present, SNN has biological basis for reference but

ANN does not, so there may be more applications of SNN in the future. At present, SNN has reached the accuracy comparable to that of deep network in many tasks, but it faces a serious problem: the time-consuming computation on neuron level and complex optimization limit their real-time application.

In this paper, we propose a CBSNN which is inspired by the curiosity-based learning mechanism in the brain. The CBSNN model can improve the accuracy and greatly reduce the computation of traditional SNN simultaneously. During the learning process, instead of feeding all data to the network, our model focuses more on mining difficult samples which is based on the novelty estimation. And in order to avoid the overfitting of the novel samples and forgetting of the learned samples, the CBSNN retrains all samples periodically. Finally, the CBSNN achieves comparable performance with the previous state-of-the-art VPSNN using just 54.95% computation of it. Similar conclusion can also be found out in other datasets, i.e., Iris, NETtalk, Fashion-MNIST, and CIFAR-10, respectively.

One of the main motivations of the paper is to dramatically decrease the training time of SNNs and make them better simulated on traditional computing systems by combining the biological plausible rules. Besides, with the development of neuroscience and physiology, more mechanisms in biological systems will be found out, which would further help SNNs on the faster processing speed and less computation cost.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: <http://yann.lecun.com/exdb/mnist/>.

AUTHOR CONTRIBUTIONS

MS and YZ designed the study, performed the experiments and the analysis, and wrote the paper. YZ and TZ were involved in problem definition, algorithm discussion, and result analysis.

FUNDING

This study was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No.XDB32070100), the Beijing Municipality of Science and Technology (Grant No. Z181100001518006), the CETC Joint

Fund (Grant No. 6141B08010103), the Major Research Program of Shandong Province (Grant No. 2018CXGC1503), the Beijing Natural Science Foundation (No. 4184103), the National Natural Science Foundation of China (No. 61806195), and the Beijing Academy of Artificial Intelligence (BAAI).

REFERENCES

- Audhkhasi, K., Rosenberg, A., Sethy, A., Ramabhadran, B., and Kingsbury, B. (2017). End-to-end asr-free keyword search from speech. *IEEE J. Select. Top. Sig. Process.* 11, 1351–1359. doi: 10.1109/JSTSP.2017.2759726
- Baranes, A. F., Oudeyer, P.-Y., and Gottlieb, J. (2014). The effects of task difficulty, novelty and the size of the search space on intrinsically motivated exploration. *Front. Neurosci.* 8:317. doi: 10.3389/fnins.2014.00317
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). “Curriculum learning,” in *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, QC: ACM), 41–48. doi: 10.1145/1553374.1553380
- Cheng, B., Wei, Y., Shi, H., Chang, S., Xiong, J., and Huang, T. S. (2018). Revisiting pre-training: an efficient training method for image classification. *arXiv:1811.09347*.
- Chernova, S., and Veloso, M. M. (2009). Interactive policy learning through confidence-based autonomy. *J. Artif. Intell. Res.* 34, 1–25. doi: 10.1613/jair.2584
- Costa, V. D., Tran, V. L., Turchi, J., and Averbeck, B. B. (2014). Dopamine modulates novelty seeking behavior during decision making. *Behav. Neurosci.* 128, 556–566. doi: 10.1037/a0037128
- 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
- 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
- Dreyer, J.-L. (2010). New insights into the roles of microRNAs in drug addiction and neuroplasticity. *Genome Med.* 2:92. doi: 10.1186/gm213
- Faraji, M., Preuschoff, K., and Gerstner, W. (2018). Balancing new against old information: the role of puzzlement surprise in learning. *Neural Comput.* 30, 34–83. doi: 10.1162/neco_a_01025
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Ann Eugen.* 7, 179–188. doi: 10.1111/j.1469-1809.1936.tb02137.x
- 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/fnirc.2015.00085
- Gerstner, W. (2001). “A framework for spiking neuron models: the spike response model,” in *Handbook of Biological Physics*, Vol. 4, eds F. Moss and S. Gielen (Elsevier), 469–516.
- Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.
- Haber, N., Mrowca, D., Wang, S., Fei-Fei, L. F., and Yamins, D. L. (2018). “Learning to play with intrinsically-motivated, self-aware agents,” in *Advances in Neural Information Processing Systems* (Montreal, QC), 8388–8399.
- He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). “Mask r-cnn,” in *Proceedings of the IEEE International Conference on Computer Vision* (Venice), 2961–2969. doi: 10.1109/ICCV.2017.322
- Hérisé, C., Khalil, R., Moftah, M., Boraud, T., Guthrie, M., and Garenne, A. (2016). Decision making under uncertainty in a spiking neural network model of the basal ganglia. *J. Integr. Neurosci.* 15, 515–538. doi: 10.1142/S021963521650028X
- Ikemoto, S. (2010). Brain reward circuitry beyond the mesolimbic dopamine system: a neurobiological theory. *Neurosci. Biobehav. Rev.* 35, 129–150. doi: 10.1016/j.neubiorev.2010.02.001
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440
- Kang, M. J., Hsu, M., Krajchich, I. M., Loewenstein, G., McClure, S. M., Wang, J. T., et al. (2009). The wick in the candle of learning: epistemic curiosity activates reward circuitry and enhances memory. *Psychol. Sci.* 20, 963–973. doi: 10.1111/j.1467-9280.2009.02402.x
- Khalil, R., Moftah, M. Z., Landry, M., and Moustafa, A. A. (2017a). Chapter 23: Models of dynamical synapses and cortical development. *Comput. Models Brain Behav.* 321–331. doi: 10.1002/9781119159193.ch23
- Khalil, R., Moftah, M. Z., and Moustafa, A. A. (2017b). The effects of dynamical synapses on firing rate activity: a spiking neural network model. *Eur. J. Neurosci.* 46, 2445–2470. doi: 10.1111/ejn.13712
- 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. U.S.A.* 114, 3521–3526. doi: 10.1073/pnas.1611835114
- Konyushkova, K., Sznitman, R., and Fua, P. (2017). “Learning active learning from data,” in *Advances in Neural Information Processing Systems* (Long Beach, CA), 4225–4235.
- Krizhevsky, A., and Hinton, G. (2009). *Learning Multiple Layers of Features From Tiny Images*. Technical report, Citeseer.
- LeCun, Y. (1998). *The MNIST Database of Handwritten Digits*. Available online at: <http://yann.lecun.com/exdb/mnist/>
- 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
- Maaten, L. V. D., and Hinton, G. (2008). Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, 2579–2605.
- Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: a review. *Neural Netw.* 113, 54–7. doi: 10.1016/j.neunet.2019.01.012
- Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). “Curiosity-driven exploration by self-supervised prediction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (Honolulu, HI), 16–17.
- Pitti, A., Mori, H., Kouzuma, S., and Kuniyoshi, Y. (2009). Contingency perception and agency measure in visuo-motor spiking neural networks. *IEEE Trans. Auton. Ment. Dev.* 1, 86–97. doi: 10.1109/TAMD.2009.2021506
- Sahay, A., Scobie, K. N., Hill, A. S., O’Carroll, C. M., Kheirbek, M. A., Burghardt, N. S., et al. (2011). Increasing adult hippocampal neurogenesis is sufficient to improve pattern separation. *Nature* 472:466. doi: 10.1038/nature09817
- Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., et al. (2019). “Episodic curiosity through reachability,” in *International Conference on Learning Representations* (New Orleans, LA).
- Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11:24. doi: 10.3389/fncom.2017.00024
- Schmidhuber, J. (1991a). “Adaptive confidence and adaptive curiosity,” in *Institut für Informatik, Technische Universität München, Arcisstr. 21, 800 München 2* (Citeseer).
- Schmidhuber, J. (1991b). “A possibility for implementing curiosity and boredom in model-building neural controllers,” in *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior* (Cambridge, MA: MIT Press), 222–227.
- Sejnowski, T. J., and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Comput. Syst.* 1, 145–168.
- Shrestha, S. B., and Orchard, G. (2018). “Slayer: spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems* (Montreal, QC), 1412–1421.
- Soltoggio, A., Stanley, K. O., and Risi, S. (2018). Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Netw.* 108, 48–67. doi: 10.1016/j.neunet.2018.07.013
- 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

ACKNOWLEDGMENTS

The authors appreciate Feifei Zhao and Dongcheng Zhao for valuable discussions. The authors would like to thank all the reviewers for their help in shaping and refining the paper.

- Ugur, E., Dogar, M. R., Cakmak, M., and Sahin, E. (2007). "Curiosity-driven learning of traversability affordance on a mobile robot," in *IEEE 6th International Conference on Development and Learning* (London, UK), 13–18.
- Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*.
- Yogatama, D., Miao, Y., Melis, G., Ling, W., Kuncoro, A., Dyer, C., et al. (2018). "Memory architectures in recurrent neural network language models," in *International Conference on Learning Representations* (Vancouver, BC).
- Zenke, F., Agnes, E. J., and Gerstner, W. (2015). Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *Nat. Commun.* 6, 6922–6922. doi: 10.1038/ncomms7922
- Zhang, T., Zeng, Y., Zhao, D., and Shi, M. (2018a). "A plasticity-centric approach to train the non-differential spiking neural networks," in *Thirty-Second AAAI Conference on Artificial Intelligence* (New Orleans, LA).
- Zhang, T., Zeng, Y., Zhao, D., Wang, L., Zhao, Y., and Xu, B. (2016). "Hmsnn: hippocampus inspired memory spiking neural network," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (Budapest: IEEE), 002301–002306.
- Zhang, T., Zeng, Y., Zhao, D., and Xu, B. (2018b). "Brain-inspired balanced tuning for spiking neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence* (Stockholm), 1653–1659. doi: 10.24963/ijcai.2018/229
- Zhao, F., Zeng, Y., and Xu, B. (2018). A brain-inspired decision-making spiking neural network and its application in unmanned aerial vehicle. *Front. Neurobot.* 12:56. doi: 10.3389/fnbot.2018.00056
- Zhou, Z., Shin, J., Zhang, L., Gurudu, S., Gotway, M., and Liang, J. (2017). "Fine-tuning convolutional neural networks for biomedical image analysis: actively and incrementally," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 7340–7351. doi: 10.1109/CVPR.2017.506
- Zola, S., Manzanares, C., Levey, A., and Lah, J. (2011). Predicting the onset of Alzheimer's disease with a behavioral task. *Alzheimer's Dement.* 7:S549. doi: 10.1016/j.jalz.2011.05.1549

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.

Copyright © 2020 Shi, Zhang and Zeng. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Corrigendum: A Curiosity-Based Learning Method for Spiking Neural Networks

Mengting Shi^{1,2†}, Tielin Zhang^{1†} and Yi Zeng^{1,2,3,4*†}

¹ Research Center for Brain-inspired Intelligence, Institute of Automation, Chinese Academy of Sciences, Beijing, China, ² University of Chinese Academy of Sciences, Beijing, China, ³ Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Shanghai, China, ⁴ National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China

Keywords: curiosity, spiking neural network, novelty, STDP, voltage-driven plasticity-centric SNN

A Corrigendum on

A Curiosity-Based Learning Method for Spiking Neural Networks

by Shi, M., Zhang, T., and Zeng, Y. (2020). *Front. Comput. Neurosci.* 14:7. doi: 10.3389/fncom.2020.00007

OPEN ACCESS

Edited and reviewed by:

Huajin Tang,
Zhejiang University, China

*Correspondence:

Yi Zeng
yi.zeng@ia.ac.cn

[†]These authors have contributed
equally to this work

Received: 21 February 2020

Accepted: 23 March 2020

Published: 15 April 2020

Citation:

Shi M, Zhang T and Zeng Y (2020)
Corrigendum: A Curiosity-Based
Learning Method for Spiking Neural
Networks.
Front. Comput. Neurosci. 14:28.
doi: 10.3389/fncom.2020.00028

In the original article, there was an error. In the original main text, there was an inaccurate statement sentence the result of NETalk in Table 3.

A correction has been made to **Experiments, The validation of CBSNN on other datasets:**

- NETalk (Sejnowski and Rosenberg, 1987) is usually used for speech generation, consisting 5,033 training and 500 test. The input is a string of letters with fixed length of 7, which is encoded into 189 dimensions (each character has a 27 length one-hot vector). The output is 26 dimensions which represent 72 phonetic principles. For this mapping task with strong global regularities, VPSNN reaches 0.8680 accuracy. Although CBSNN is only slightly higher than VPSNN, it saves about half of the computation cost.

The authors apologize for this error and state that this does not change the scientific conclusions of the article in any way. The original article has been updated.

REFERENCES

Sejnowski, T. J., and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce english text. *Compl. Syst.* 1, 145–168.

Copyright © 2020 Shi, Zhang and Zeng. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Probabilistic Circuits for Autonomous Learning: A Simulation Study

Jan Kaiser*, Rafatul Faria, Kerem Y. Camsari and Supriyo Datta

Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Modern machine learning is based on powerful algorithms running on digital computing platforms and there is great interest in accelerating the learning process and making it more energy efficient. In this paper we present a fully autonomous probabilistic circuit for fast and efficient learning that makes no use of digital computing. Specifically we use SPICE simulations to demonstrate a clockless autonomous circuit where the required synaptic weights are read out in the form of analog voltages. This allows us to demonstrate a circuit that can be built with existing technology to emulate the Boltzmann machine learning algorithm based on gradient optimization of the maximum likelihood function. Such autonomous circuits could be particularly of interest as standalone learning devices in the context of mobile and edge computing.

Keywords: on-device learning, Boltzmann machine algorithm, probabilistic computing, magnetic tunnel junction (MTJ), machine learning, analog circuit

OPEN ACCESS

Edited by:

Lei Deng,

University of California, Santa Barbara,
United States

Reviewed by:

Youhui Zhang,

Tsinghua University, China

Jeongjun Lee,

Texas A&M University, United States

*Correspondence:

Jan Kaiser

kaiser32@purdue.edu

Received: 25 November 2019

Accepted: 03 February 2020

Published: 25 February 2020

Citation:

Kaiser J, Faria R, Camsari KY and Datta S (2020) Probabilistic Circuits for Autonomous Learning: A Simulation Study. *Front. Comput. Neurosci.* 14:14. doi: 10.3389/fncom.2020.00014

1. INTRODUCTION

Machine learning, inference, and many other emerging applications (Schuman et al., 2017) make use of stochastic neural networks comprising (1) a binary stochastic neuron (BSN) (Ackley et al., 1985; Neal, 1992) and (2) a synapse that constructs the inputs I_i to the i th BSN from the outputs m_j of all other BSNs.

The output m_i of the i th BSN fluctuates between +1 and -1 with a probability controlled by its input

$$m_i(t + \tau_N) = \text{sgn} [\tanh (I_i(t)) - r] \quad (1)$$

where r represents a random number in the range $[-1, +1]$, and τ_N is the time it takes for a neuron to provide a stochastic output m_i in accordance with a new input I_i ¹.

Usually the synaptic function, $I_i(\{m\})$ is linear and is defined by a set of weights W_{ij} such that

$$I_i(t + \tau_S) = \sum_j W_{ij} m_j(t) \quad (2)$$

where τ_S is the time it takes to recompute the inputs $\{I\}$ everytime the outputs $\{m\}$ change. Typically Equations (1), (2) are implemented in software, often with special accelerators for the synaptic function using GPU/TPUs (Schmidhuber, 2015; Jouppi, 2016).

The time constants τ_N and τ_S are not important when Equations (1) and (2) are implemented on a digital computer using a clock to ensure that neurons are updated sequentially and the synapse is updated between any two updates. But they play an important role in clockless operation of autonomous hardware that makes use of the natural physics of specific systems to implement Equations (1) and (2) approximately. A key advantage of using BSNs is that Equation (1) can be

¹Equation (1) can be written in binary notation with a unit step function and a sigmoid function.

implemented compactly using stochastic magnetic tunnel junctions (MTJs) as shown in Camsari et al. (2017a,b), while resistive or capacitive crossbars can implement Equation (2) (Hassan et al., 2019a). It has been shown that such hardware implementations can operate autonomously without clocks, if the BSN operates slower than the synapse, that is, if $\tau_N \gg \tau_S$ shown by Sutton et al. (2019).

Stochastic neural networks defined by Equations (1) and (2) can be used for inference whereby the weights W_{ij} are designed such that the system has a very high probability of visiting configurations defined by $\{m\} = \{v\}_n$, where $\{v\}_n$ represents a specified set of patterns. However, the most challenging and time-consuming part of implementing a neural network is not the inference function, but the learning required to determine the correct weights W_{ij} for a given application. This is commonly done using powerful cloud-based processors and there is great interest in accelerating the learning process and making it more energy efficient so that it can become a routine part of mobile and edge computing.

In this paper we present a new approach to the problem of fast and efficient learning that makes no use of digital computing at all. Instead it makes use of the natural physics of a fully autonomous probabilistic circuit composed of standard electronic components like resistors, capacitors, and transistors along with stochastic MTJs.

We focus on a fully visible Boltzmann machine (FVBM), a form of stochastic recurrent neural network, for which the most common learning algorithm is based on the gradient ascent approach to optimize the maximum likelihood function (Carreira-Perpinan and Hinton, 2005; Koller and Friedman, 2009). We use a slightly simplified version of this approach, whereby the weights are changed incrementally according to

$$W_{ij}(t + \Delta t) = W_{ij}(t) + \epsilon[v_i v_j - m_i m_j - \lambda W_{ij}(t)]$$

where ϵ is the learning parameter and λ is the regularization parameter (Ng, 2004). The term $v_i v_j$ is the correlation between the i th and the j th entry of the training vector $\{v\}_n$. The term $m_i m_j$ corresponds to the sampled correlation taken from the model's distribution. The advantage of this network topology is that the learning rule is local since it only requires information of the two neurons i and j connected by weight W_{ij} . In addition, the learning rule can tolerate stochasticity for example in the form of sampling noise which makes it an attractive algorithm to use for hardware machine learning (Carreira-Perpinan and Hinton, 2005; Fischer and Igel, 2014; Ernoul et al., 2019).

For our autonomous operation we replace the equation above with its continuous time version (τ_L : learning time constant)

$$\frac{dW_{ij}}{dt} = \frac{v_i v_j - m_i m_j - \lambda W_{ij}}{\tau_L} \quad (3)$$

which we translate into an RC circuit by associating W_{ij} with the voltage on a capacitor C driven by a voltage source ($V_{v,ij} - V_{m,ij}$) with a series resistance R (Figure 1):

$$C \frac{dV_{ij}}{dt} = \frac{V_{v,ij} - V_{m,ij} - V_{ij}}{R} \quad (4)$$

with $v_i v_j = V_{v,ij}/(V_{DD}/2)$ and $m_i m_j = V_{m,ij}/(V_{DD}/2)$. From Figure 1 and comparing Equations (3), (4) it is easy to see how the weights and the learning and regularization parameters are mapped into circuit elements: $W_{ij} = A_v V_{ij}/V_0$, $\lambda = V_0/(A_v V_{DD}/2)$, and $\tau_L = \lambda RC$ where A_v is the voltage gain of OP3 in Figure 1 and V_0 is the reference voltage of the BSN. For proper operation the learning time scale τ_L has to be much larger than the neuron time τ_N to be able to collect enough statistics throughout the learning process.

A key element of this approach is the representation of the weights W with voltages rather than with programmable resistances for which memristors and other technologies are still in development (Li et al., 2018b). By contrast the charging of capacitors is a textbook phenomenon, allowing us to design a learning circuit that can be built today with established technology. The idea of using capacitor voltages to represent weights in neural networks has been presented by several authors for different network topologies in analog learning circuits (Schneider and Card, 1993; Card et al., 1994; Kim et al., 2017; Sung et al., 2018). The use of capacitors has the advantage of having a high level of linearity and symmetry for the weight updates during the training process (Li et al., 2018a).

In section 2, we will describe such a learning circuit that emulates Equations (1)–(3). The training images or patterns $\{v\}_n$ are fed in as electrical signals into the input terminals, and the synaptic weights W_{ij} can then be read out in the form of voltages from the output terminals. Alternatively the values can be stored in a non-volatile memory from which they can subsequently be read and used for inference. In section 3, we will present SPICE simulations demonstrating the operation of this autonomous learning circuit.

2. METHODS

The autonomous learning circuit has three parts where each part represents one of the three Equations (1)–(3). On the left hand side of Figure 1, the training data is fed into the circuit by supplying a voltage $V_{v,ij}$ which is given by the i th entry of the bipolar training vector v_i multiplied by the j th entry of the training vector v_j and scaled by the supply voltage $V_{DD}/2$. The training vectors can be fed in sequentially or as an average of all training vectors. The weight voltage V_{ij} across capacitor C follows Equation (4) where $V_{v,ij}$ is compared to voltage $V_{m,ij}$ which represents correlation of the outputs of BSNs m_i and m_j . Voltage $V_{m,ij}$ is computed in the circuit by using an XNOR gate that is connected to the output of BSN i and BSN j . The synapse in the center of the circuit connects weight voltages to neurons according to Equation (2). Voltage V_{ij} has to be multiplied by 1 or -1 depending on the current value of m_j . This is accomplished by using a switch which connects either the positive or the negative node of V_{ij} to the operational amplifiers OP1 and OP2. Here, OP1 accumulates all negative contributions and OP2 accumulates all positive contributions of the synaptic function. The differential amplifier OP3 takes the difference between the output voltages of OP2 and OP1 and amplifies the voltage by amplification factor A_v . This voltage conversion is used to control the voltage level of V_{ij} in relation to the input voltage of each BSN. The voltage level at the input of the BSN is fixed by the reference voltage

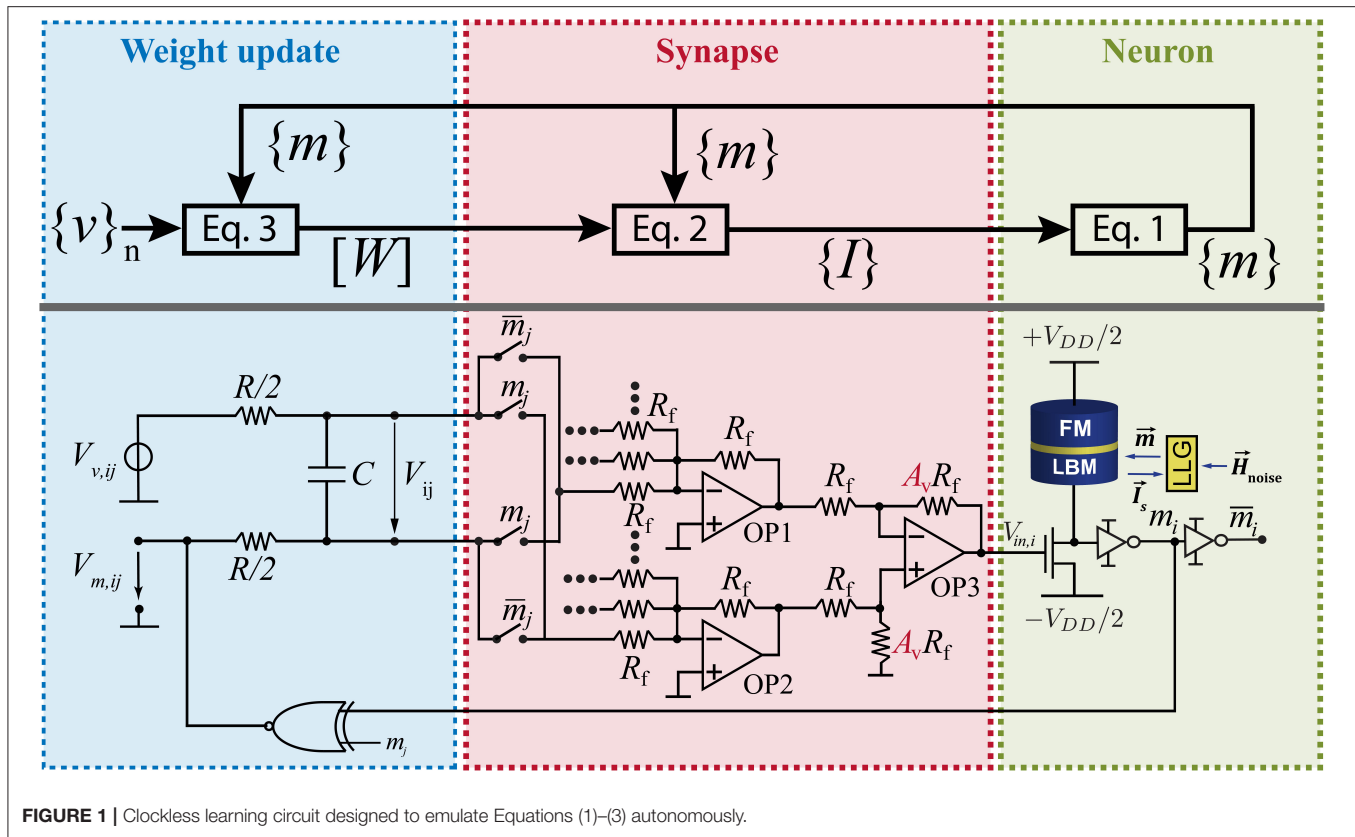


FIGURE 1 | Clockless learning circuit designed to emulate Equations (1)–(3) autonomously.

of the BSN which is V_0 . However, the voltage level of V_{ij} can be adjusted and utilized to adjust the regularization parameter λ in the learning rule (Equation 3). The functionality of the BSN is described by Equation (1) where the dimensionless input is given by $I_i(t) = V_{i,in}(t)/V_0$. This relates the voltage V_{ij} to the dimensionless weight by $W_{ij} = A_v V_{ij}/V_0$. The hardware implementation of the BSN uses a stochastic MTJ in series with a transistor as presented by Camsari et al. (2017b). Due to thermal fluctuations of the low-barrier magnet (LBM) of the MTJ the output voltage of the MTJ fluctuates randomly but with the right statistics given by Equation (1). The time dynamics of the LBM can be obtained by solving the stochastic Landau-Lifshitz-Gilbert (LLG) equation. Due to the fast thermal fluctuations of the LBM in the MTJ, Equation (1) can be evaluated on a subnanosecond timescale leading to fast generation of samples (Hassan et al., 2019b; Kaiser et al., 2019b).

Figure 1 just shows the hardware implementation of one weight and one BSN. The size of the whole circuit depends on the size of the training vector N . For every entry of the training vector one BSN is needed. The number of weights which is the number of RC-circuits is given by $N(N - 1)/2$ where every connection between BSNs is assumed to be reciprocal. To learn biases another N RC-circuits are needed.

The learning process is captured by Equations (3) and (4). The whole learning process has similarity with the software implementation of persistent contrastive divergence (PCD) (Tieleman, 2008) since the circuit takes samples from the model's

distribution ($V_{m,ij}$) and compares it to the target distribution ($V_{v,ij}$) without reinitializing the Markov Chain after a weight update. During the learning process voltage V_{ij} reaches a constant average value where $\frac{dV_{ij}}{dt} \approx 0$. This voltage $V_{ij} = V_{ij,learned}$ corresponds to the learned weight.

For inference the capacitor C is replaced by a voltage source of voltage $V_{ij,learned}$. Consequently, the autonomous circuit will compute the desired functionality given by the training vectors. In general, training and inference have to be performed on identical hardware in order to learn around variations (see **Supplementary Material** for more details). It is important to note that in inference mode this circuit can be used for optimization by performing electrical annealing. This is done by increasing all weight voltages V_{ij} by the same factor over time. In this way the ground state of a Hamiltonian like the Ising Hamiltonian can be found (Sutton et al., 2017; Camsari et al., 2019).

3. RESULTS

In this section the autonomous learning circuit in **Figure 1** is simulated in SPICE. We show how the proposed circuit can be used for both inference and learning. As examples, we demonstrate the learning on a full adder (FA) and on 5×3 digit images. The BSN models are simulated in the framework developed by Camsari et al. (2015). For all SPICE simulations the

TABLE 1 | Truth table of a full adder.

A v_1	B v_2	C _{in} v_3	S v_4	C _{out} v_5	Dec	P_{Ideal}
-1	-1	-1	-1	-1	0	0.125
-1	-1	1	1	-1	6	0.125
-1	1	-1	1	-1	10	0.125
-1	1	1	-1	1	13	0.125
1	-1	-1	1	-1	18	0.125
1	-1	1	-1	1	21	0.125
1	1	-1	-1	1	25	0.125
1	1	1	1	1	31	0.125

Every 0 in the binary representation of the full adder is replaced by -1 in the bipolar representation. "Dec" represents the decimal conversion of each line. P_{Ideal} is the ideal probability distribution were every line's probability is $p=1/8=0.125$.

following parameters are used for the stochastic MTJ in the BSN implementation: Saturation magnetization $M_S = 1,100$ emu/cc, LBM diameter $D = 22$ nm, LBM thickness $l = 2$ nm, TMR = 110%, damping coefficient $\alpha = 0.01$, temperature $T = 300$ K and demagnetization field $H_D = 4\pi M_S$ with $V = (D/2)^2 \pi l$. For the transistors, 14 nm HP-FinFET Predictive Technology Models (PTM)² are used with fin number $fin = 1$ for the inverters and $fin = 2$ for XNOR-gates. Ideal operational amplifiers and switches are used in the synapse. The characteristic time of the BSNs τ_N is in the order of 100 ps (Hassan et al., 2019b) and much larger than the time it takes for the synaptic connections, namely the resistors and operational amplifiers, to propagate BSN outputs to neighboring inputs. It has to be noted that in principle other hardware implementations of the synapse for computing Equation (2) could be utilized as long as the condition $\tau_N \gg \tau_S$ is satisfied.

3.1. Learning Addition

As first training example, we use the probability distribution of a full adder. The FA has 5 nodes and 10 weights that have to be learned. In the case of the FA training, no biases are needed. The probability distribution of a full adder with bipolar variables is shown in **Table 1**. To learn this distribution the correlation terms $v_i v_j$ in the learning rule have to be fed into the voltage node $V_{v,ij}$. The correlation is dependent on what training vector/truth table line is fed in. For the second line of the truth table for example $v_1 v_2 = -1 \cdot -1 = 1$ and $v_1 v_3 = -1 \cdot 1 = -1$ with A being the first node, B the second node and so on. In **Figure 2B** the correlation $v_1 v_5$ is shown. For the sequential case the value of $v_1 v_5$ is obtained by circling through all lines of the truth table where each training vector is shown for 1 ns. A and C_{out} in **Table 1** only differ in the fourth and fifth line for which $v_1 v_5 = -1$. For all other cases $v_1 v_5 = 1$. The average of all lines is shown as red solid line. **Figure 2A** shows the weight voltage V_{ij} with $i = 1$ and $j = 5$ for FA learning and the first 1,000 ns of training. The following learning parameters have been used for the FA: $\tau_L = 62.5$ ns where $C = 1$ nF and $R = 5$ k Ω , $A_v = 10$, and $R_f = 1$ M Ω . This choice of learning parameters

²<http://ptm.asu.edu/>

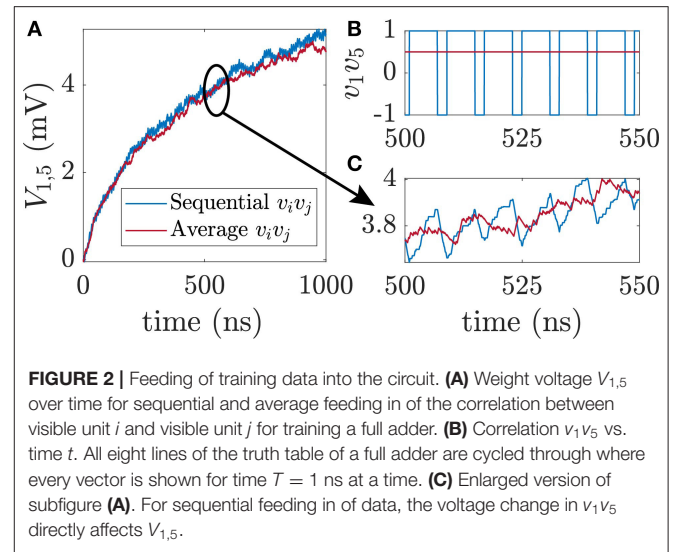
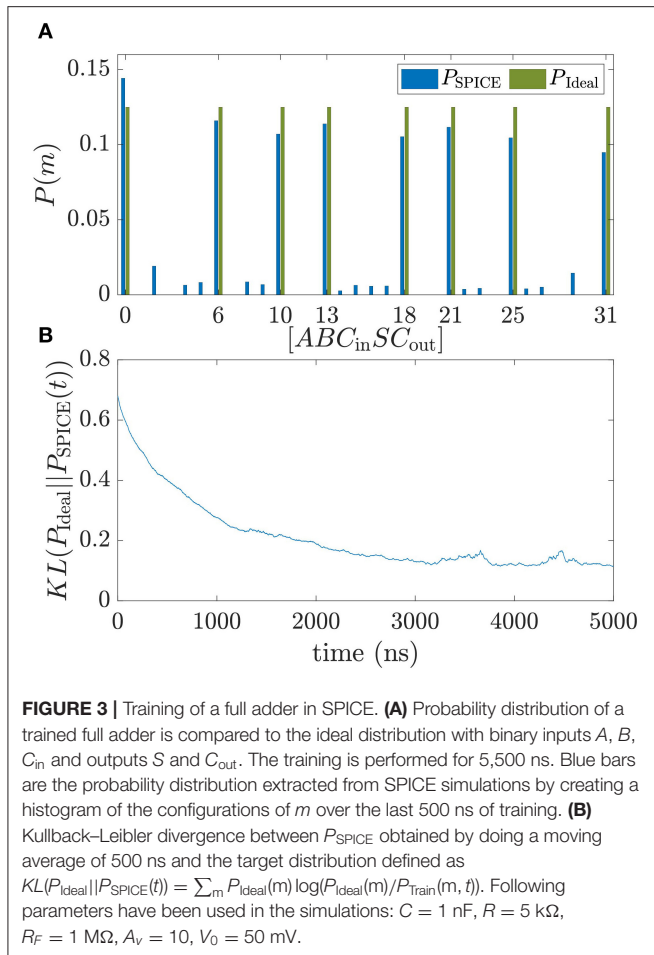


FIGURE 2 | Feeding of training data into the circuit. **(A)** Weight voltage $V_{1,5}$ over time for sequential and average feeding in of the correlation between visible unit i and visible unit j for training a full adder. **(B)** Correlation $v_1 v_5$ vs. time t . All eight lines of the truth table of a full adder are cycled through where every vector is shown for time $T = 1$ ns at a time. **(C)** Enlarged version of subfigure **(A)**. For sequential feeding in of data, the voltage change in $v_1 v_5$ directly affects $V_{1,5}$.

ensures that $\tau_L \gg \tau_N$. Due to the averaging effect of the RC-circuit both sequential and average feeding of the training vector result in similar learning behavior as long as the RC-constant is much larger than the timescale of sequential feeding. **Figure 2C** shows the enlarged version of **Figure 2A**. For the sequential feeding, voltage $V_{1,5}$ changes substantially every time $v_1 v_5$ switches to -1.

At the start of training all weight voltages are initialized to 0 V and the probability distribution is uniform. The training is performed for 5,500 ns. In **Figure 3A** the ideal probability distribution of the FA P_{Ideal} is shown together with the normalized histogram P_{SPICE} of the sampled BSN configurations taken from the last 500 ns of learning and compared to the ideal distribution P_{Ideal} . The training vector is fed in as an average. For P_{SPICE} the eight trained configurations of **Table 1** are the dominant peaks. To monitor the training process, the Kullback-Leibner divergence between the trained and the ideal probability distribution $KL(P_{\text{Ideal}}||P_{\text{SPICE}}(t))$ is plotted as a function of training time t in **Figure 3B** where $P_{\text{SPICE}}(t)$ is the normalized histogram taken over 500 ns. P_{SPICE} at $t = 0$ corresponds to the histogram taken from $t = 0$ to $t = 500$ ns. During training the KL divergence decreases over time until it reaches a constant value at about 0.1. It has to be noted that after the weight matrix is learned correctly for a fully visible Boltzmann machine, the KL divergence can be reduced further by increasing all weights uniformly by a factor I_0 which corresponds to inverse temperature of the Boltzmann machine (Aarts and Korst, 1989). **Figure 3** shows that the probability distribution of a FA can be learned very fast with the proposed autonomous learning circuit. In addition, the learning performance is robust when components of the circuit are subject to variation. In the **Supplementary Material**, additional figures of the learning performance are shown when the diameter of the magnet and the resistances of the RC-circuits are subject to variation. The robustness against variations can be explained by the fact that the circuit can learn around variations. BSNs using LBMs under variations

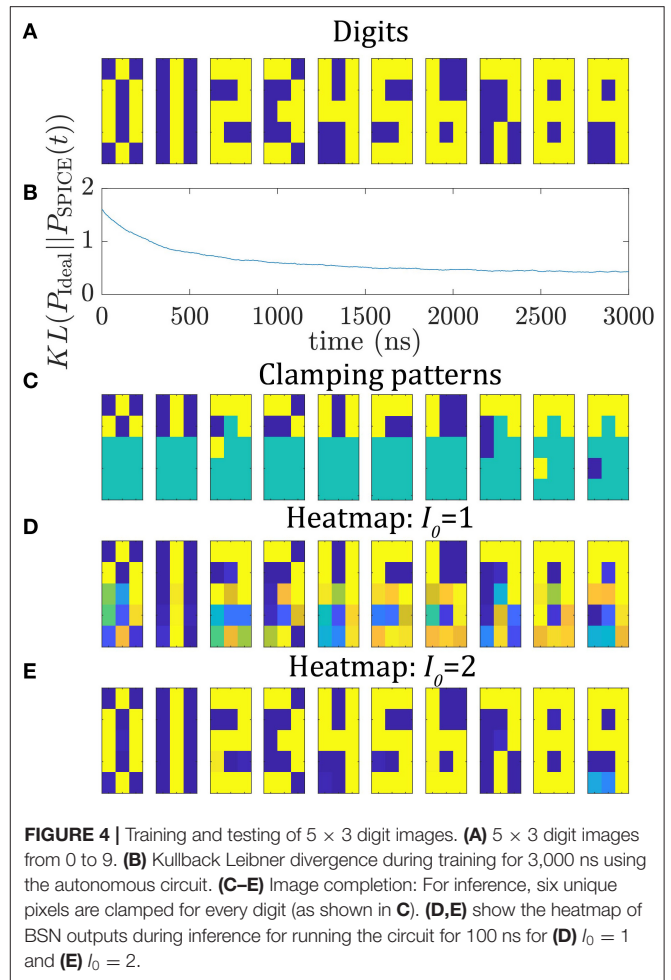


have also been analyzed by Abeer and Bandyopadhyay (2019) and Drobitch and Bandyopadhyay (2019).

3.2. Learning Image Completion

As second example, the circuit is utilized to train 10 5×3 pixel digit images shown in **Figure 4A**. Here, 105 reciprocal weights and 15 biases have to be learned. The network is trained for 3,000 ns and the bipolar training data is fed in as average of the 10 $v_i v_j$ terms for every digit. The same learning parameters as in the previous section are used here. In **Figure 4B**, the KL divergence is shown as a function of time between the SPICE histogram and the ideal probability distribution where the ideal distribution has 10 peaks with each peak being 10% for each digit. Most of the learning happens in the first 1,500 ns of training, however, the KL divergence still reduces slightly during the later parts of learning. After 3,000 ns the KL divergence reaches a value of around 0.5.

For inference we replace the capacitor by a voltage source where every voltage is given by the previously learned voltage V_{ij} . The circuit is run for 10 instances where every instance has a unique clamping pattern of 6 pixels representing one of the 10 digits. The clamped inputs are shown in **Figure 4C**. The input of a clamped BSN is set to $\pm V_{DD}/2$. Each instance is run



for 100 ns and the outputs of the BSNs are monitored. The BSNs fluctuate between the configurations given by the learned probability distribution. In **Figure 4D**, the heat map of the output of the BSNs is shown. For every digit the most likely configuration is given by the trained digit image. To illustrate this point, the amount of BSN fluctuations is reduced by increasing the learned weight voltages by a factor of $I_0 = 2$. The circuit is again run in inference mode for 100 ns with the same clamping patterns. In **Figure 4E** the heatmap is shown. The circuit locks into the learned digit configuration. This shows that in inference mode the circuit can be utilized for image completion.

4. DISCUSSION

In this paper we have presented a framework for mapping a continuous version of Boltzmann machine learning rule (Equation 3) to a clockless autonomous circuit. We have shown full SPICE simulations to demonstrate the feasibility of this circuit running without any digital component with the learning parameters set by circuit parameters. Due to the fast BSN operation, samples are drawn at subnanosecond speeds leading to fast learning, as such the learning speed should

be at least multiple orders of magnitudes faster compared to other computing platforms (Adachi and Henderson, 2015; Korenkevych et al., 2016; Terenin et al., 2019). The advantage of this autonomous architecture is that it produces random numbers naturally and does not rely on pseudo random number generators like linear-feedback shift register (LFSRs) (which are for example used in Bojnordi and Ipek, 2016). These LFSRs have overhead and are not as compact and efficient as the hardware BSN used in this paper. As shown by Borders et al. (2019), typical LFSRs need about 10x more energy per flip and more than 100x more area than an MTJ-based BSN. Another advantage of this approach is that the interfacing with digital hardware only needs to be performed after the learning has been completed. Hence, no expensive analog-to-digital conversion has to be performed during learning. We believe this approach could be extended to other energy based machine learning algorithms like equilibrium propagation introduced by Scellier and Bengio (2017) to design autonomous circuits. Such standalone learning devices could be particularly of interest in the context of mobile and edge computing.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

REFERENCES

- Aarts, E., and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY: John Wiley & Sons, Inc.
- Abeed, M. A., and Bandyopadhyay, S. (2019). Low energy barrier nanomagnet design for binary stochastic neurons: design challenges for real nanomagnets with fabrication defects. *IEEE Magn. Lett.* 10, 1–5. doi: 10.1109/LMAG.2019.2929484
- Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cogn. Sci.* 9, 147–169. Available online at: <https://www.sciencedirect.com/science/article/abs/pii/S0364021385800124>
- Adachi, S. H., and Henderson, M. P. (2015). Application of quantum annealing to training of deep neural networks. *arXiv*: 1510.06356. Available online at: <https://arxiv.org/abs/1510.06356>
- Bojnordi, M. N., and Ipek, E. (2016). “Memristive Boltzmann machine: a hardware accelerator for combinatorial optimization and deep learning,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (IEEE), 1–13. Available online at: <https://ieeexplore.ieee.org/abstract/document/7446049>
- Borders, W. A., Pervaiz, A. Z., Fukami, S., Camsari, K. Y., Ohno, H., and Datta, S. (2019). Integer factorization using stochastic magnetic tunnel junctions. *Nature* 573, 390–393. doi: 10.1038/s41586-019-1557-9
- Camsari, K. Y., Chowdhury, S., and Datta, S. (2019). Scalable emulation of sign-problem-free Hamiltonians with room-temperature SpS -bits. *Phys. Rev. Appl.* 12:034061. doi: 10.1103/PhysRevApplied.12.034061
- Camsari, K. Y., Faria, R., Sutton, B. M., and Datta, S. (2017a). Stochastic p-bits for invertible logic. *Phys. Rev. X* 7:031014. doi: 10.1103/PhysRevX.7.031014
- Camsari, K. Y., Ganguly, S., and Datta, S. (2015). Modular approach to spintronics. *Sci. Rep.* 5:10571. doi: 10.1038/srep10571
- Camsari, K. Y., Salahuddin, S., and Datta, S. (2017b). Implementing p-bits with embedded MTJ. *IEEE Elect. Device Lett.* 38, 1767–1770. doi: 10.1109/LED.2017.2768321
- Card, H. C., Schneider, C. R., and Schneider, R. S. (1994). Learning capacitive weights in analog CMOS neural networks. *J. VLSI Signal Process.* 8, 209–225. doi: 10.1007/BF02106447
- Carreira-Perpinan, M. A., and Hinton, G. E. (2005). “On contrastive divergence learning,” in *Aistats*, Vol. 10, 33–40. Available online at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.8829&rep=rep1&type=pdf#page=42>
- Drobitch, J. L., and Bandyopadhyay, S. (2019). Reliability and scalability of p-bits implemented with low energy barrier nanomagnets. *IEEE Magn. Lett.* 10, 1–4. doi: 10.1109/LMAG.2019.2956913
- Ernoul, M., Grollier, J., and Querlioz, D. (2019). Using memristors for robust local learning of hardware restricted Boltzmann machines. *Sci. Rep.* 9:1851. doi: 10.1038/s41598-018-38181-3
- Fischer, A., and Igel, C. (2014). Training restricted Boltzmann machines: an introduction. *Pattern Recogn.* 47, 25–39. doi: 10.1016/j.patcog.2013.05.025
- Hassan, O., Camsari, K. Y., and Datta, S. (2019a). Voltage-driven building block for hardware belief networks. *IEEE Design Test* 36, 15–21. doi: 10.1109/MDAT.2019.2897964
- Hassan, O., Faria, R., Camsari, K. Y., Sun, J. Z., and Datta, S. (2019b). Low-barrier magnet design for efficient hardware binary stochastic neurons. *IEEE Magn. Lett.* 10, 1–5. doi: 10.1109/LMAG.2019.2910787
- Jouppi, N. (2016). *Google Supercharges Machine Learning Tasks With TPU Custom Chip*. Google Cloud Platform Blog. Google. Available online at: <https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip>
- Kaiser, J., Faria, R., Camsari, K. Y., and Datta, S. (2019a). Probabilistic circuits for autonomous learning: a simulation study. *arXiv [Preprint]*. *arXiv*: 1910.06288. Available online at: <https://arxiv.org/abs/1910.06288>
- Kaiser, J., Rustagi, A., Camsari, K. Y., Sun, J. Z., Datta, S., and Upadhyaya, P. (2019b). Subnanosecond fluctuations in low-barrier nanomagnets. *Phys. Rev. Appl.* 12:054056. doi: 10.1103/PhysRevApplied.12.054056
- Kim, S., Gokmen, T., Lee, H. M., and Haensch, W. E. (2017). “Analog CMOS-based resistive processing unit for deep neural network training,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (IEEE),

AUTHOR CONTRIBUTIONS

JK and SD wrote the paper. JK performed the simulations. RF helped setting up the simulations. KC developed the simulation modules for the BSN. All authors discussed the results and helped refine the manuscript.

FUNDING

This work was supported in part by ASCENT, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. KC gratefully acknowledges support from Center for Science of Information (CSOI), an NSF Science and Technology Center, under grant CCF-0939370.

ACKNOWLEDGMENTS

This manuscript has been released as a preprint at arXiv (Kaiser et al., 2019a).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fncom.2020.00014/full#supplementary-material>

- 422–425. Available online at: <https://ieeexplore.ieee.org/abstract/document/8052950>
- Koller, D., and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press. Available online at: <https://mitpress.mit.edu/books/probabilistic-graphical-models>
- Korenkevych, D., Xue, Y., Bian, Z., Chudak, F., Macready, W. G., Rolfe, J., et al. (2016). Benchmarking quantum hardware for training of fully visible Boltzmann machines. *arXiv:1611.04528*. Available online at: <https://arxiv.org/abs/1611.04528>
- Li, Y., Kim, S., Sun, X., Solomon, P., Gokmen, T., Tsai, H., et al. (2018a). “Capacitor-based cross-point array for analog neural network with record symmetry and linearity,” in *2018 IEEE Symposium on VLSI Technology*, 25–26. Available online at: <https://ieeexplore.ieee.org/document/8510648>
- Li, Y., Wang, Z., Midya, R., Xia, Q., and Yang, J. J. (2018b). Review of memristor devices in neuromorphic computing: materials sciences and device challenges. *J. Phys. D Appl. Phys.* 51:503002. doi: 10.1088/1361-6463/aade3f
- Neal, R. M. (1992). Connectionist learning of belief networks. *Artif. Intell.* 56, 71–113. doi: 10.1016/0004-3702(92)90065-6
- Ng, A. Y. (2004). “Feature selection, L_1 vs. L_2 regularization, and rotational invariance,” in *Twenty-First International Conference on Machine Learning - ICML '04* (Banff, AB: ACM Press), 78.
- Scellier, B., and Bengio, Y. (2017). Equilibrium propagation: bridging the gap between energy-based models and backpropagation. *Front. Comput. Neurosci.* 11:24. doi: 10.3389/fncom.2017.00024
- Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Netw.* 61, 85–117. doi: 10.1016/j.neunet.2014.09.003
- Schneider, C. R., and Card, H. C. (1993). Analog CMOS deterministic Boltzmann circuits. *IEEE J. Solid State Circ.* 28, 907–914. doi: 10.1109/4.231327
- 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. *arXiv [Preprint]*. *arXiv:1705.06963 [cs]*. Available online at: <https://arxiv.org/abs/1705.06963>
- Sung, C., Hwang, H., and Yoo, I. K. (2018). Perspective: a review on memristive hardware for neuromorphic computation. *J. Appl. Phys.* 124:151903. doi: 10.1063/1.5037835
- Sutton, B., Camsari, K. Y., Behin-Aein, B., and Datta, S. (2017). Intrinsic optimization using stochastic nanomagnets. *Sci. Rep.* 7:44370. doi: 10.1038/srep44370
- Sutton, B., Faria, R., Ghantasala, L. A., Camsari, K. Y., and Datta, S. (2019). Autonomous probabilistic coprocessing with petaflips per second. *arXiv [Preprint]*. *arXiv:1907.09664*. Available online at: <https://arxiv.org/abs/1907.09664>
- Terenin, A., Dong, S., and Draper, D. (2019). GPU-accelerated Gibbs sampling: a case study of the Horseshoe Probit model. *Stat. Comput.* 29, 301–310. doi: 10.1007/s11222-018-9809-3
- Tieleman, T. (2008). “Training restricted Boltzmann machines using approximations to the likelihood gradient,” in *Proceedings of the 25th International Conference on Machine Learning - ICML '08* (Helsinki: ACM Press), 1064–1071. doi: 10.1145/1390156.1390290

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.

Copyright © 2020 Kaiser, Faria, Camsari and Datta. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Spike-Train Level Direct Feedback Alignment: Sidestepping Backpropagation for On-Chip Training of Spiking Neural Nets

Jeongjun Lee¹, Renqian Zhang², Wenrui Zhang¹, Yu Liu² and Peng Li^{1*}

¹ Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States, ² Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, United States

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

Priyadarshini Panda,
Yale University, United States
Junxiu Liu,
Ulster University, United Kingdom

*Correspondence:

Peng Li
lip@ucsb.edu

Specialty section:

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

Received: 07 September 2020

Accepted: 04 February 2020

Published: 13 March 2020

Citation:

Lee J, Zhang R, Zhang W, Liu Y and
Li P (2020) Spike-Train Level Direct
Feedback Alignment: Sidestepping
Backpropagation for On-Chip Training
of Spiking Neural Nets.
Front. Neurosci. 14:143.
doi: 10.3389/fnins.2020.00143

Spiking neural networks (SNNs) present a promising computing model and enable bio-plausible information processing and event-driven based ultra-low power neuromorphic hardware. However, training SNNs to reach the same performances of conventional deep artificial neural networks (ANNs), particularly with error backpropagation (BP) algorithms, poses a significant challenge due to inherent complex dynamics and non-differentiable spike activities of spiking neurons. In this paper, we present the first study on realizing competitive spike-train level backpropagation (BP) like algorithms to enable on-chip training of SNNs. We propose a novel spike-train level direct feedback alignment (ST-DFA) algorithm, which is much more bio-plausible and hardware friendly than BP. Algorithm and hardware co-optimization and efficient online neural signal computation are explored for on-chip implementation of ST-DFA. On the Xilinx ZC706 FPGA board, the proposed hardware-efficient ST-DFA shows excellent performance vs. overhead tradeoffs for real-world speech and image classification applications. SNN neural processors with on-chip ST-DFA training show competitive classification accuracy of 96.27% for the MNIST dataset with 4× input resolution reduction and 84.88% for the challenging 16-speaker TI46 speech corpus, respectively. Compared to the hardware implementation of the state-of-the-art BP algorithm HM2-BP, the design of the proposed ST-DFA reduces functional resources by 76.7% and backward training latency by 31.6% while gracefully trading off classification performance.

Keywords: spiking neural networks, backpropagation, on-chip training, hardware neural processor, FPGA

1. INTRODUCTION

As a brain-inspired computational model, spiking neural networks (SNNs) have gathered significant research interests during recent years. The spike-based operational principles of SNNs support a variety of information coding schemes including temporal codes and have rendered energy-efficient VLSI neuromorphic chips, such as IBM's TrueNorth (Akopyan et al., 2015) and Intel's Loihi (Davies et al., 2018). Despite the recent progresses in SNNs and neuromorphic processor designs, fully leveraging the theoretical computing advantages of SNNs over traditional artificial neural networks (ANNs) (Maass, 1997) to achieve the state-of-the-art performance for real-world applications remains challenging. One chief difficulty here lies in training of SNNs in terms of achievable performance and computational complexity.

In terms of the learning algorithm of SNNs, there are several algorithms, such as Spike-timing-dependent plasticity (STDP)/binary/probabilistic and error backpropagation (BP). Each algorithm has gathered significant research interests with the advantages of each algorithm. For example, STDP mimics biological behavior using the timing between pre- and post-synaptic spikes, and BP has shown the state-of-the-art performance in ANNs, implying its potential to be used in SNNs for achieving excellent accuracy. While the above algorithms provide a rich set of learning mechanisms that can be explored, as of now, SNNs exploiting a non-BP algorithm, such as a STDP/binary/probabilistic method have demonstrated limited success in competitive real-world applications. Recent advances in BP have provided algorithms for overcoming non-differentiability of spike events and capturing temporal dynamics, and have made it possible to achieve the state-of-the-art performances among many other algorithms.

Inspired by the success of BP and its variants, such as stochastic gradient descent in training conventional ANNs (Rumelhart et al., 1988a), various SNN BP methods have emerged, aiming at attaining the same level of performance (Bohte et al., 2002; Lee et al., 2016; Jin et al., 2018; Wu et al., 2018; Chankyu et al., 2019; Panda et al., 2019). The major challenges in BP training of SNNs stem from the non-differentiability of spike events and temporal dynamics that prevent straightforward derivative computation. SpikeProp (Bohte et al., 2002) is the first BP algorithm to train SNNs by BP. However, SpikeProp is limited to single-spike training for learning simple functions like XOR. Lee et al. (2016) proposes a BP algorithm which differentiates neuron's membrane potential instead of discrete output spikes. Wu et al. (2018) improves Lee et al. (2016) by capturing temporal effects with backpropagation through time (BPTT) (Werbos, 1990). However, the error gradient is still computed by differentiating the membrane potential, leading to inconsistency w.r.t the rate-coded loss function. More recently, Panda et al. (2019) provides the hybrid neural network architecture for approximate gradient descent (AGD) training methodology achieving good accuracy with large datasets, such as Imagenet, and Chankyu et al. (2019) proposes differentiable activation for leaky integrate-and-fire (LIF) spiking neurons using a spike-based BP algorithm achieving good classification accuracies with various datasets.

This paper is motivated by one of these approaches (i.e., Jin et al., 2018), which showed a spike-train level BP algorithm that achieves the state-of-the-art performance on SNNs. Jin et al. (2018) proposes a hybrid macro/micro level backpropagation (HM2-BP) algorithm for training multi-layer SNNs, which addresses the aforementioned issues. HM2-BP precisely captures the temporal behavior of the SNN at the microscopic level and directly computes the gradient of the rate-coded loss function w.r.t tunable parameters. As a result, HM2-BP demonstrates the state-of-the-art learning performances on widely adopted SNN benchmarks, such as MNIST (LeCun et al., 1998) and Neuromorphic-MNIST (N-MNIST) (Orchard et al., 2015), outperforming all other existing BP algorithms based on the leaky integrate-and-fire model.

While achieving excellent results, the aforementioned SNN BP algorithms are hampered by several limitations. The error signal is propagated backward layer by layer through weights symmetric to the feed-forward weights. This is considered not biologically-plausible. Furthermore, BP algorithms involve complex layer-by-layer backward computations, which is expensive to implement on-chip and introduces high training latency. For instance, while HM2-BP improves the scalability of BPTT (Wu et al., 2018) by operating on the spike-train level, i.e., application of BP does not discretize time, it still involves complex computations and its latency in the backward phase is proportional to network depth.

This work aims to answer the following questions: (1) Can biologically plausible mechanisms be developed to sidestep complex BP algorithms while delivering competitive performance? (2) Can such mechanisms be leveraged for efficient on-chip training of multi-layer SNNs?

We are motivated by the recent direct feedback alignment (DFA) method developed for conventional ANNs (Nøkland, 2016), where the error is more biologically-plausibly fed back to each hidden layer through fixed random feedback connections directly from the output layer, reducing a bulk of the BP complexity. Furthermore, DFA can be performed for all hidden layers concurrently, reducing the backward phase latency.

By extending the DFA concept proposed by Nøkland (2016) for SNNs, we significantly reduce hardware overhead and latency of the network, while maintaining the advantage of a well-defined BP-like algorithm in terms of accuracy. Although many algorithms, such as gradient descent (GD), AGD, and BP, have been proposed for SNNs, this is the first work presenting algorithm-hardware co-optimization and demonstrating the realization of DFA for SNNs with significantly reduced hardware cost while maintaining competitive accuracies for image/speech recognition tasks. The main contributions of this work are:

- We demonstrate the *first* direct feedback alignment algorithm for training multi-layer SNNs by extending the DFA concept developed for conventional ANNs;
- Our spiking DFA algorithm is embodied at the spike-train level, dubbed ST-DFA, to further improve scalability by avoiding involved error feedback over time;
- We perform algorithm-hardware co-optimization and demonstrate the *first* hardware realization of DFA for SNNs with significantly reduced hardware overhead, energy dissipation, and latency while achieving competitive performances for image/speech recognition tasks.

On the Xilinx ZC706 FPGA board, the proposed ST-DFA with optimized implementation shows excellent cost-effectiveness for on-chip SNN training. Hardware SNNs with ST-DFA deliver competitive accuracy of 96.27% for the MNIST (LeCun et al., 1998) with 4× input resolution reduction and 84.88% for the challenging 16-speaker TI46 (Lieberman et al., 1991) speech corpus, respectively. Compared to the hardware implementation of the state-of-the-art BP algorithm HM2-BP, the design of the proposed ST-DFA reduces functional resources by 76.7% and backward training latency by 31.6% while gracefully trading off classification performance.

2. MATERIALS AND METHODS

2.1. Background

2.1.1. Direct Feedback Alignment

Backpropagation (BP) has been widely applied to train neural networks. It is based upon computing a global error at the output layer and then propagating the error signal to hidden neurons layer by layer. During this process, the errors of a preceding layer are multiplied with a weight matrix that is completely symmetric to the one for the feed-forward connections. This fact is not considered biologically plausible. A recent discovery called Feedback Alignment (FA) (Lillicrap et al., 2016) demonstrates that the weights used for propagating the error layer by layer need not be symmetric to the weights used for forward propagation to achieve good performance. The feedback weight matrix can be randomly generated and then stay unchanged since the network can learn how to make feedback useful through training. Neftci et al. (2017) applies FA for training SNNs.

A more disruptive approach called Direct Feedback Alignment (DFA) is proposed in DNNs (Nøkland, 2016). DFA is compared with BP in **Figure 1**. Unlike propagating the error back layer by layer in BP and FA, DFA feeds back the error through fixed random feedback connections directly from the output layer to each hidden layer, eliminating the need for layer-by-layer error backpropagation or feedback. DFA is considered more biologically plausible because the error is generated almost completely local with no long backpropagation/feed back train and symmetric weights are not required. Nøkland (2016) shows that for conventional multi-layer ANNs like DNNs, the use of DFA can achieve competitive results with insignificant performance drops when compared with the state-of-the-art BP methods.

In this paper, we extend the DFA for conventional ANNs (Nøkland, 2016) for SNNs. To the best of our knowledge, this is the first work applying DFA to SNNs. Furthermore, our DFA approach, dubbed ST-DFA, operates on the spike-train level, hence offering improved scalability in both space (network depth) and time.

2.1.2. Spike-Train Level Post-synaptic Potential

Before describing the proposed ST-DFA in section 2.2, we present the concept of Spike-train Level Post-synaptic Potential (S-PSP) that is behind the spike-train level computation of ST-DFA.

The widely adopted leaky integrate-and-fire (LIF) model for spiking neurons is given by (Gerstner and Kistler, 2002):

$$\tau_m \frac{du_i(t)}{dt} = -u_i(t) + R \alpha_i(t), \quad (1)$$

with

$$\tau_s \frac{d\alpha_i(t)}{dt} = -\alpha_i(t) + \sum_j w_{ij} \sum_{t_j^{(f)}} D\left(t - t_j^{(f)}\right), \quad (2)$$

where $u_i(t)$ is the membrane potential of the neuron i , $\alpha_i(t)$ is the first order synaptic model with time constant τ_s , and τ_m is the time constant of membrane potential with value $\tau_m = RC$. R and C are the effective leaky resistance and effective membrane

capacitance. w_{ij} is the weight of the synapse from the pre-synaptic neuron j to the neuron i . $t_j^{(f)}$ denotes a particular firing time of the neuron j . $D(t)$ is the Dirac delta function. R is set to 1 since it can be absorbed into synaptic weights.

Integrating (1) and (2) gives the spike response model (SRM) (Jin et al., 2018):

$$u_i(t) = \sum_j w_{ij} \sum_{t_j^{(f)}} \epsilon\left(t - \hat{t}_i^{(f)}, t - t_j^{(f)}\right), \quad (3)$$

where $\hat{t}_i^{(f)}$ denotes the last firing time of the neuron i . $\epsilon(s, t)$ specifies the normalized time course of the *post-synaptic potential* evoked by a single firing spike of the pre-synaptic neuron:

$$\epsilon(s, t) = \frac{1}{C} \int_0^s \exp\left(-\frac{t'}{\tau_m}\right) \alpha_i(t - t') dt'. \quad (4)$$

Integrating (4) gives:

$$\epsilon(s, t) = \frac{e^{(-\max(t-s, 0)/\tau_s)}}{1 - \frac{\tau_s}{\tau_m}} \left[e^{\left(-\frac{\min(s, t)}{\tau_m}\right)} - e^{\left(-\frac{\min(s, t)}{\tau_s}\right)} \right] H(s)H(t), \quad (5)$$

where $H(t)$ is the Heaviside step function.

The sum of the (normalized) post-synaptic potential of the neuron i evaluated right before all the neuron i 's firing times evoked by the spike train of the pre-synaptic neuron j defines the (normalized) spike-train level post-synaptic potential (S-PSP) e_{ij} , which is given by:

$$e_{ij} = \sum_{t_i^{(f)}} \sum_{t_j^{(f)}} \epsilon(t_i^{(f)} - \hat{t}_i^{(f)}, t_i^{(f)} - t_j^{(f)}). \quad (6)$$

S-PSP specifies the aggregated effect of the spike train of the pre-synaptic neuron j on the membrane potential of the post-synaptic neuron i , providing a basis for relating firing counts to spike events.

Summing the weighted S-PSPs from all pre-synaptic neurons of the neuron i gives the total post-synaptic potential (T-PSP) a_i , which is directly correlated to the neuron i 's firing count o_i through the firing threshold voltage v :

$$a_i = \sum_j w_{ij} e_{ij}. \quad o_i = g(a_i) \approx \frac{a_i}{v} \quad (7)$$

2.2. Proposed Spike-Train Level Direct Feedback Alignment (ST-DFA)

2.2.1. Proposed ST-DFA Algorithm

For a conventional (non-spiking) ANN, the squared error for one training example can be defined at the output layer by:

$$E = \frac{1}{2} \|\mathbf{o} - \mathbf{y}\|_2^2, \quad (8)$$

where \mathbf{y} and \mathbf{o} are vectors specifying the desired output (label) and the actual output, respectively. The output o_i of each neuron

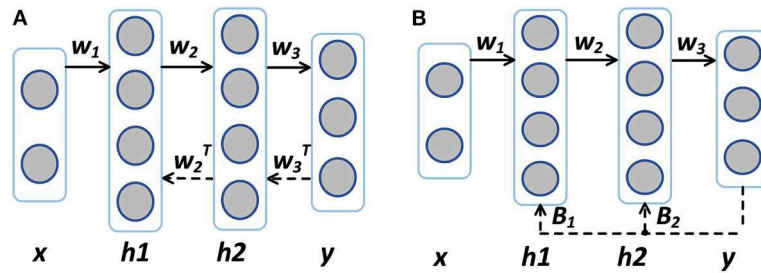


FIGURE 1 | (A) Backpropagation (BP) vs. **(B)** Direct feedback alignment (DFA). Solid arrows indicate feedforward paths and dashed arrows indicate feedback paths. The feedback matrices B_1 and B_2 need not be symmetric to W_2 or W_3 .

i is determined by the activation function ϕ_i : $o_i = \phi_i(\sum_j w_{ij}x_j)$, where x_j is the input value from the presynaptic neuron j and w_{ij} is the weight between the neurons j and i .

The well-known BP algorithm for an ANN (Rumelhart et al., 1988b), which is ubiquitously used in deep learning, is:

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}^k} = \eta \delta_i^k \phi_j^{k-1} \quad (9)$$

$$\delta_i^k = \begin{cases} o_i - y_i & \text{for output layer,} \\ \phi_i^{k+1} \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li}^{k+1} & \text{for hidden layers,} \end{cases}$$

where η is the learning rate, δ_i^k the error for the i th neuron of the k th layer, r^k the number of neurons in the k th layer.

It has been demonstrated recently that training SNNs using BP with respect to a rate-coded loss function has produced highly competitive performances (Lee et al., 2016; Jin et al., 2018; Wu et al., 2018). Rate-coded loss functions are also adopted for our ST-DFA. Different from BP, the proposed ST-DFA algorithm for SNNs computes each error δ by direct feedback from the output layer on the spike-train level, giving to the following update rule:

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} = \eta \delta_i^k e_{ij}^k, \quad (10)$$

$$\delta_i^k = \begin{cases} \frac{o_i^o - y_i^o}{v} & \text{for output layer,} \\ \sum_{l=1}^{r^o} \delta_l^o b_{li}^k & \text{for hidden layers,} \end{cases}$$

where η is the learning rate, δ_i^k the error of the neuron i in the k th hidden layer, e_{ij}^k the S-PSP from the neuron i to neuron j , o_i^o the actual firing count of neuron i in the output layer, y_i^o the desired firing count for the neuron i , v the firing threshold, r^o the number of neurons in the output layer, δ_l^o the error of the neuron l in the output layer, and b_{li}^k the value of the fixed random feedback.

The last equation of (10) is based on the concept of DFA. As in **Figure 2**, with ST-DFA, the output layer is fully connected to each hidden layer through a different matrix which is called the random feedback matrix B . The weights (values) in these matrices are randomly generated and then stay fixed. The error vector δ^k of the hidden layer k is directly obtained from the error vector of the output layer δ^o and the random feedback matrix

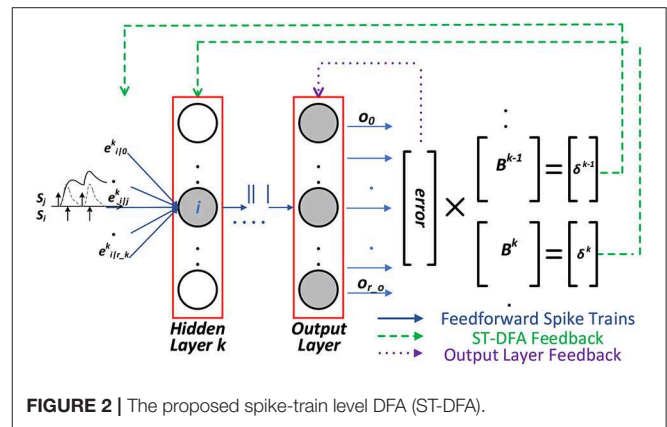


FIGURE 2 | The proposed spike-train level DFA (ST-DFA).

B^k as: $\delta^k = B^k \times \delta^o$. The detailed derivation of ST-DFA is introduced next.

2.2.2. Derivation of ST-DFA

Similar to (8) and using (7), we define the rate-coded loss function as:

$$E = \frac{1}{2} \|\mathbf{o} - \mathbf{y}\|_2^2 = \frac{1}{2} \left\| \frac{\mathbf{a}}{v} - \mathbf{y} \right\|_2^2, \quad (11)$$

where \mathbf{y} , \mathbf{o} , and \mathbf{a} are vectors specifying the desired firing counts (label), the actual firing counts, and the T-PSP of the output neurons, respectively. Differentiating the loss function with respect to each trainable weight w_{ij} leads to:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_i^k} \frac{\partial a_i^k}{\partial w_{ij}} = \delta_i^k \frac{\partial a_i^k}{\partial w_{ij}}, \quad (12)$$

where a_i^k is the T-PSP of the neuron i in the k th layer.

It is instrumental to note that each S-PSP e_{ij} depends on both rate and temporal information of the pre/post-spike trains, i.e., e_{ij} depends on the pre/post-synaptic firing counts o_i and o_j and pre/post-synaptic firing times $\mathbf{t}_j^{(f)}$ and $\mathbf{t}_i^{(f)}$:

$$e_{ij} = f(o_j, o_i, \mathbf{t}_j^{(f)}, \mathbf{t}_i^{(f)}). \quad (13)$$

For the i th output neuron, δ_i^o can be obtained from (12) and (7):

$$\delta_i^o = \frac{\partial E}{\partial a_i^o} = (o_i - y_i) \frac{\partial o_i}{\partial a_i} = \frac{o_i - y_i}{v}. \quad (14)$$

For each i th neuron in the hidden layer k , δ_i^k is derived from the chain rule based on (7):

$$\begin{aligned} \delta_i^k &= \frac{\partial E}{\partial a_i^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_i^k} = \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_i^k} \\ &= \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li}^{k+1} \frac{\partial e_{li}^{k+1}}{\partial a_i^k}. \end{aligned} \quad (15)$$

The first key development in ST-DFA is that the way in which the error δ_i^k is calculated in each hidden layer changes from $\sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li}^{k+1} \frac{\partial e_{li}^{k+1}}{\partial a_i^k}$ to $\sum_{l=1}^{r^o} \delta_l^o d_{li}^k \frac{\partial e_{li}^{k+1}}{\partial a_i^k}$, where d_{li}^k is the direct feedback alignment from the output neuron l to the hidden layer neuron i . d_{li}^k is a randomized and fixed value. In this process, we replace the w_{li}^{k+1} from $(k+1)$ th layer to k th layer in (15) by d_{li}^k , leading to:

$$\delta_i^k = \delta_l^o d_{li}^k \frac{\partial e_{li}^{k+1}}{\partial a_i^k}. \quad (16)$$

As such, the error δ^k of each hidden neuron is directly determined by the output layer error vector δ^o rather than by the error vector δ^{k+1} of the next layer.

Moreover, we have the following key observation. In (16), since d_{li}^k is randomly generated, $\frac{\partial e_{li}^{k+1}}{\partial a_i^k}$ can be absorbed into d_{li}^k to further simplify ST-DFA. Denote the new DFA parameter absorbing $\frac{\partial e_{li}^{k+1}}{\partial a_i^k}$ by $b_{li}^k = d_{li}^k \frac{\partial e_{li}^{k+1}}{\partial a_i^k}$, the simplified error computation becomes:

$$\delta_i^k = \begin{cases} \frac{o_i - y_i}{v} & \text{for output layer,} \\ \sum_{l=1}^{r^o} \delta_l^o b_{li}^k & \text{for hidden layers,} \end{cases} \quad (17)$$

where b_{li}^k is one entry of the random feedback matrix \mathbf{B} in **Figure 2**.

Thus, ST-DFA reduces the computational complexity by not only avoiding layer-by-layer propagation but also the additional simplification via the use of b_{li}^k .

2.2.3. Simplification for Hardware Friendliness

The last term on the right-hand side of (12) differentiates the total post-synaptic potential (T-PSP) a_i^k . Considering (7), it can be written as:

$$\begin{aligned} \frac{\partial a_i^k}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \left(\sum_{j=1}^{r^{k-1}} w_{ij} e_{ij}^k \right) = e_{ij}^k + \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{il}^k}{\partial o_i^k} \frac{\partial o_i^k}{\partial w_{ij}} \\ &= e_{ij}^k + \frac{e_{ij}^k}{v} \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{il}^k}{\partial o_i^k}. \end{aligned} \quad (18)$$

The exact evaluation of the above expression requires multiple additions, multiplications, and divisions, introducing high hardware overhead and additional latency.

The first term e_{ij}^k on the right-hand side of (18) can be interpreted as the direct influence exerted on the T-PSP a_i^k by changing the synaptic weight w_{ij} as seen from (7). The second term $\frac{e_{ij}^k}{v} \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{il}^k}{\partial o_i^k}$ comes from the fact that changing the weight w_{ij} leads to variation in the post-synaptic spike train. Thus, the S-PSP e_{il}^k to the neuron i also varies as it depends on the firing times of the post-synaptic neuron. Nevertheless, we have observed that the first term dominates the second term. By dropping the second term, we reach the final hardware-friendly ST-DFA algorithm of (10), which also maintains good performance.

In comparison, the spike-train level BP algorithm HM2-BP is (Jin et al., 2018):

$$\begin{aligned} \Delta w_{ij} &= \eta \delta_i^k e_{ij}^k \left(1 + \frac{1}{v} \sum_{l=1}^{r^{k-1}} w_{il} \frac{\partial e_{il}^k}{\partial o_i^k} \right), \\ \delta_i^k &= \begin{cases} \frac{o_i - y_i}{v} & \text{for output layer,} \\ \frac{1}{v} \sum_{l=1}^{r^{k+1}} \delta_l^{k+1} w_{li} \frac{\partial e_{li}^{k+1}}{\partial o_i^k} & \text{for hidden layers.} \end{cases} \end{aligned} \quad (19)$$

While HM2-BP delivers the state-of-the-art performance, it would be very costly to implement on hardware if ever feasible.

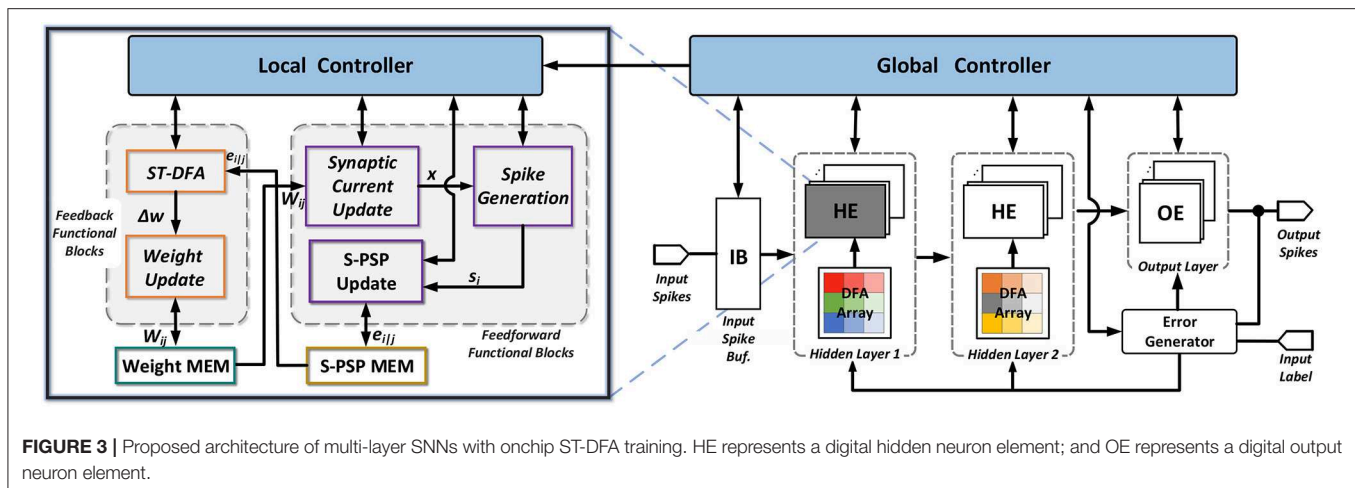
In all, compared to HM2-BP in (19), ST-DFA in (10) is much more hardware friendly. With ST-DFA, direct error feedback to each hidden layer is accomplished without layer-by-layer back propagation while HM2 requires high-resolution multiplications with the transpose of the forward weights and other expensive operations layer by layer. In the next section, we efficiently realize the ST-DFA algorithm on digital hardware.

2.3. SNN Accelerators With ST-DFA On-Chip Training

2.3.1. Architecture

Figure 3 shows the architecture of the proposed multi-layer feed-forward spiking neural processors with the proposed ST-DFA on-chip training. Only two hidden layers are shown for illustration purpose. Architecturally, the processor is comprised of an input spike buffer feeding multiple hidden layers composed of hidden neuron elements (HEs). The last hidden layer connects to the output layer which consists of a set of output neuron elements (OEs). A modular design approach is taken where each spiking neuron is implemented in the form of HE or OE. As such, a proper number of HEs and OEs can be instantiated to form a multi-layer SNN with arbitrary depth and width.

Both inference and training are supported. Training over an input example splits into two phases: forward pass and backward pass. The computation of S-PSPs required for ST-DFA training are computed in an online manner in the forward pass of training. The remaining computations of the forward pass are identical to those performed in inference. To support ST-DFA training, the error generator utilizes an array of subtractors to



compute the difference between the actual OE output spike counts with expected ones (label). At each hidden layer, this output-layer error vector is multiplied with the associated ST-DFA random feedback matrix inside each layer to allow weight updates performed by each neuron.

- On-chip training

For each training example, the forward and backward passes of the training are controlled by a global controller (FSM) as shown in **Figure 3**. Neurons at the same layer process information in parallel to exploit the inherent parallelism of the hardware SNN processor architecture. In the forward pass and at each biological time step, layers are activated by the global controller one at a time from the input to the output. After output spikes are generated for the current time step, the global controller pushes the training forward to the next time step. This process repeats until the current training example has been entirely learned by the network. Then, the backward pass starts, in which the first step is to calculate the output error δ_i^o in (10). After that, all hidden layers start to perform ST-DFA for weight updating at the same time. The weight update latency of each hidden layer may be different due to the differences in the number of input synaptic connections (i.e., the preceding layer width). After all hidden layers finish ST-DFA weight updates, the training process moves onto the next training example.

- Neuron unit design

Each HE or OE contains several functional blocks categorized into feed-forward functional blocks and feedback functional blocks as shown in **Figure 3**. OEs are identical to HEs except that no ST-DFA module is included since the error δ_i^k defined for output neurons is computed by the Error Generator module. Each neuron unit contains two memory modules that store the synaptic weights and all its spike-train level post-synaptic potentials (S-PSPs), respectively. We implement the weight memory with block RAM (BRAM) and the S-PSP memory with a 2-D array of flip flops (FFs) on the FPGA. A neuron-level local controller (FSM) controls the detailed inference/training steps. The local controller also communicates with the global

controller for synchronizing processes between different layers and inference/training stages.

In the forward pass of training, first, the synaptic current x through each synapse is calculated, followed by the spike-train level post-synaptic potential (S-PSP) update for the same synapse. The synaptic current update and the S-PSP update modules shown in **Figure 3** are shared by all input synapses. Hence, processing of all synapses are done in series. After all synaptic responses are generated, the spike generation module calculates the neuron's membrane potential and makes the firing decision based on the leaky integrate-and-fire (LIF) spiking neuron model. In the backward pass of training, the ST-DFA module implements the proposed on-chip ST-DFA training algorithm, the output of which is then fed to the weight update module. Finally, the corresponding synaptic weight is updated and stored back to the weight memory. Similar to the feedforward blocks, the feedback functional modules are also shared among all input synapses.

2.3.2. Efficient On-Chip S-PSP Calculation

One important component in the proposed ST-DFA algorithm is the spike-train level post-synaptic potential (S-PSP), e_{ij} , in (10). As demonstrated in (6), by definition, e_{ij} is the effect of all firing events of the pre-synaptic neuron j on the post-synaptic neuron i . However, direct implementation of (6) on hardware is very costly; all firing events of the pre- and post-synaptic neurons need to be stored and excessive multiplication, division and exponentiation operations are involved, incurring much logic complexity and memory usage.

Instead, we propose an online S-PSP calculation approach with dramatically reduced hardware overhead. Rather than recording all firing events of the two neurons and computing e_{ij} at once in the backward pass, in the forward pass we accumulate and update e_{ij} at the arrival of each firing event and store the updated e_{ij} in the S-PSP memory of each neuron element.

Inspecting (3) and (6) reveals that e_{ij} is the normalized (by weight) of the contribution from the post-synaptic neuron j to the aggregated membrane potential of the post-synaptic neuron i . While the aggregated post-synaptic membrane potential is effectively tracked by the LIF model, each individual contribution

e_{ij} to it can be accumulated exactly using the following equations:

$$\begin{aligned}\tau_s \frac{p_{ij}(t)}{dt} &= -p_{ij}(t) + \sum_{t_j^{(f)}} D(t - t_j^{(f)}), \\ \tau_m \frac{q_{ij}(t)}{dt} &= -q_{ij}(t) + p_{ij}(t), \\ e_{ij}(t) &= \sum_{t_i^{(f)}} q_{ij}(t_i^{(f)}),\end{aligned}\quad (20)$$

where $p_{ij}(t)$ is the (normalized) synaptic input from the neuron j to neuron i , which is part of (2), and $q_{ij}(t)$ is interpreted as the (normalized) post-synaptic membrane voltage contribution from the neuron j to neuron i , which shall be reset to zero when the neuron i fires at a particular firing time $t_i^{(f)}$.

The hardware realization of (20) is based on discretizing it using the first-order Euler method with a fixed stepsize:

$$\begin{aligned}q_{ij}[t+1] &= (1 - \frac{1}{\tau_m})q_{ij}[t] + p_{ij}[t+1] \\ p_{ij}[t+1] &= (1 - \frac{1}{\tau_s})p_{ij}[t] + \frac{1}{\tau_s} \sum_{t_j^{(f)}} D_n(t - t_j^{(f)}) \\ \begin{cases} e_{ij}[t+1] += q_{ij}[t+1] \\ q_{ij}[t+1] = 0 \end{cases} & \text{if } t+1 = t_i^{(f)},\end{aligned}\quad (21)$$

where $D_n(\cdot)$ is the unit sample function and we have abused the notation by using t and $t+1$ to indicate a discrete time step and the step after that.

(21) allows e_{ij} to be accumulated in an online manner with great hardware efficiency and its implementation is shown in **Figure 4**. At each time step, we first update the value of p_{ij} , followed by the updates of q_{ij} and e_{ij} , controlled by the FSM states of the local controller shown in **Figure 3**. The shaded blocks in **Figure 4** are registers used to store the current-time variable values. We set both decay constants τ_s and τ_m to be a power of 2 such that multiplications/divisions are realized efficiently using shift operations. The updated e_{ij} is stored in the S-PSP memory and retrieved by the ST-DFA module during the backward training pass.

2.3.3. Efficient On-Chip ST-DFA Implementation

Figure 5 depicts the ST-DFA module in hidden neurons shown in **Figure 3**. As in (10), for each hidden neuron i , the inner product between the error vector δ_i^o from the output layer and the i th column of the random feedback matrix B of the corresponding layer is computed. The inner product is then multiplied with e_{ij} to produce the weight update value Δw_{ij} for the j th input synapse. All these inner products for different synapses are computed in series and would result in large hardware and power overheads. Furthermore, if each entry of the feedback matrix is set to be a high-bit resolution random number, high memory usage is required for storage.

To mitigate the above design complexity, we propose a hardware-friendly realization of ST-DFA, named ST-DFA-2.

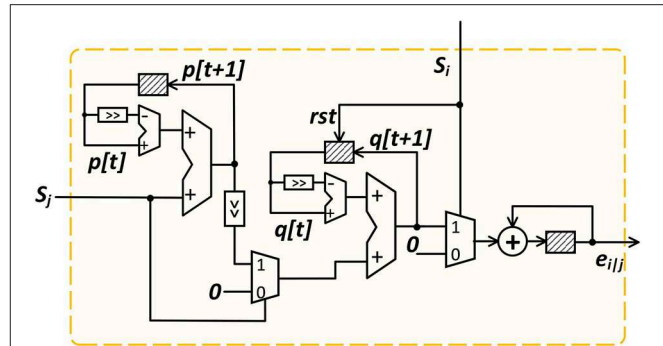


FIGURE 4 | On-line S-PSP calculation onchip.

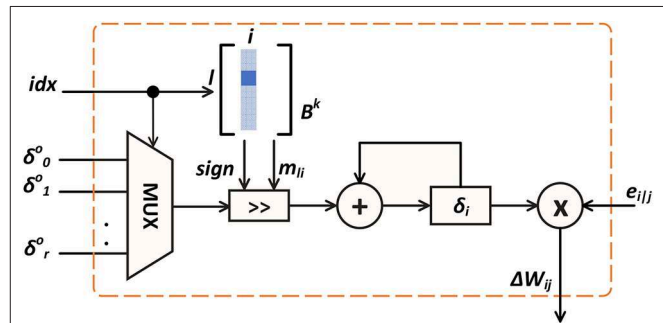


FIGURE 5 | On-chip ST-DFA weight update computation.

ST-DFA-2 is based on the key observation from extensive algorithmic experiments that the feedback matrix B need not be generated in a true random manner; setting each entry b_{li} of B to one of a small set of fixed numbers at random is sufficient for achieving good training performance. Furthermore, the set of fixed numbers can be optimized for hardware efficiency. For this, we construct this set by making each number a signed power of 2 with low-bit resolution such that the multiplications in (10) can be implemented by shift operations and storage for B is kept at minimal.

Figure 5 illustrates the computation of each weight update. The corresponding inner product is computed by accumulating the element-wise products. The idx signal selects a particular element in the error vector δ_i^o and its shift amount m_{li} , which is set by the corresponding b_{li} in the B matrix according to $|b_{li}| = 2^{m_{li}}$. If b_{li} is negative, the shift result is converted to its complement before added to δ_i . Finally, the resulting δ_i is multiplied with the S-PSP e_{ij} to get the weight update value Δw_{ij} for the current synapse.

3. RESULTS

3.1. Experimental Settings and Benchmarks

Performance evaluation is divided into two parts:

- (1) Section 3.2 devotes to evaluate the performance of proposed ST-DFA and ST-DFA-2 compared to HM2-BP only with

software simulation. The classification performances are evaluated by simulation of the digital computations with the actual bit resolutions implemented on FPGA. Major SNN variables, for example synaptic weight w , S-PSP e_{ij} and membrane potential v , are in the fixed-point representation. Each w is a signed 17-bit variable with 12-bit fractional. 11 bits are used for each unsigned variable e_{ij} with 6-bit fractional and 9 bits are used for each signed variable v with 3-bit fractional.

- (2) In section 3.3, we measure various aspects of the SNN neural processor based on the pure hardware platform (on-board measurement). We measure the performance vs. hardware overhead tradeoffs of the proposed on-chip ST-DFA training on several feed-forward SNN neural processors. Using multiple SNNs models with varying depths and widths, we demonstrate competitive performance of pure hardware (on-board) simulation. Compared to hardware implementation of HM2-BP, proposed ST-DFA significantly reduces hardware overhead which proves hardware-friendliness. FPGA prototypes of SNN neural accelerators are designed on the Xilinx ZC706 platform for performance evaluation, design overhead, and power/energy analysis.

Three datasets are employed for evaluation: MNIST (LeCun et al., 1998), N-MNIST, or the neuromorphic version of MNIST (Orchard et al., 2015), and the 16-speaker English letter subset of the TI46 speech corpus (Liberman et al., 1991). The MNIST handwritten digit dataset (LeCun et al., 1998) contains 60k training and 10k testing examples, each of which is a 28×28 grayscale image. Each pixel value of the MNIST image is converted into a spike train using Poisson sampling and the probability of spike generation is proportional to the pixel intensity. Due to the limited hardware resources available on the Xilinx Zynq ZC706 board, we crop each image to include only the 14×14 pixels around the center for FPGA evaluation.

The N-MNIST dataset (Orchard et al., 2015) is a neuromorphic version of MNIST. The static digit images of MNIST are converted into spike trains using a dynamic vision sensor (DVS) (Lichtsteiner et al., 2008) moving on a pan-tilt unit. The image is resized to 34×34 since the relative shift of images during the saccade process is required. Two kinds of spike events, ON and OFF, are recorded since the intensity can either increase or decrease. Thus, each N-MNIST image has $34 \times 34 \times 2 = 2,312$ spike sequences lasting for about 300 ms. We reduce the time resolution of the N-MNIST images by $500 \times$ to speed up the processing.

The TI46 Speech corpus (Liberman et al., 1991) contains spoken English letters from 16 speaker. There are 4,142 and 6,628 spoken English letters for training and testing, respectively. The continuous temporal speech waveforms are first preprocessed by the Lyon's ear model (Lyon, 1982) and then encoded into 78 spike trains using the BSA algorithm (Schrauwen and Van Campenhout, 2003).

Among these datasets, MNIST and TI46 are tested on both software and hardware while N-MNIST is only tested on software simulation due to that the available FPGA resources are not sufficient to support the large number of spike trains.

Moreover, to thoroughly assess the classification performance and hardware benefits of our proposed spike-train level direct feedback alignment (ST-DFA), we build multiple SNNs with different network depths and widths.

3.2. Classification Accuracies (Software Simulation)

The proposed spike-train level direct feedback alignment (ST-DFA) algorithm is inspired by the spike-train level backpropagation HM2-BP algorithm. In Jin et al. (2018), HM2-BP is compared with other state-of-the-art spiking or non-spiking BP methods, such as spike-based BP (Lee et al., 2016), STBP (Wu et al., 2018), temporal coding BP (Mostafa, 2018), and non-spiking BP (Neil et al., 2016) on MNIST and N-MNIST. Apart from its high efficiency due to the spike-train level processing, HM2-BP outperforms or is on a par with all these recently developed algorithms. For example, with a single hidden layer of 800 neurons, HM2-BP can achieve 98.93% accuracy on MNIST while Neil et al. (2016) gets up to 98.30%. HM2-BP obtains 98.88% accuracy on N-MNIST compared with 97.80% by Mostafa (2018). Moreover, HM2-BP delivers competitive performance on challenging benchmarks, such as the 16-speaker spoken English letters of TI46 Speech corpus (Liberman et al., 1991) and 47-class image recognition dataset Extended MNIST (EMNIST) (Cohen et al., 2017).

As presented in section 2.2, ST-DFA propagates the errors δ from the output layer to each hidden layer directly without layer by layer error backpropagation through symmetric weights matrices. In section 2.3.3, we further optimize ST-DFA by setting each entry of the random feedback matrix B to a power of 2, leading to the hardware-friendly ST-DFA-2 algorithm. In this work, feedback matrix entries are randomly chosen from the set $\{-4, -2, -1, 0, 1, 2, 4\}$ for ST-DFA-2.

Table 1 compares the inference accuracies of HM2-BP, ST-DFA, and ST-DFA-2 on MNIST, N-MNIST, and TI46. Compared to HM2-BP, ST-DFA, and ST-DFA-2 still maintain rather competitive performance while the low computational cost and hardware-friendliness of ST-DFA-2 translate into huge hardware resources and energy overhead savings as shown later. It shall be noted that in comparison with ST-DFA, ST-DFA-2 does not necessarily degrade performance; it can even slightly outperform ST-DFA in practice.

3.3. FPGA Hardware Evaluations (On-Board Measurement)

We build several FPGA SNN accelerators on the targeted Xilinx ZC706 platform, the sizes of which are decided considering the available resources on-chip. **Tables 2, 3** shows the resource and energy overhead as well as the inference accuracies of these SNN accelerators with on-chip ST-DFA-2. Training powers are estimated by the Xilinx Power Analyzer based on application-specific workloads. With the result of behavioral simulation using a binary-converted input data sample, the tool measures the dynamic power of neural processors clocked at 100MHz as presented in **Table 2**. Compared hardware cost in two different designs, i.e., ST-DFA-2 and HM2-BP, is shown in **Table 4**.

TABLE 1 | Inference accuracy comparison of HM2BP, ST-DFA, and ST-DFA-2 derived by software simulation.

Dataset	Learning rule and network structure	Accuracy (%)
MNIST	HM2-BP: 784-800-10	98.93
MNIST	ST-DFA: 784-800-10	98.64
MNIST	ST-DFA-2: 784-800-10	98.74
N-MNIST	HM2-BP: 2312-800-10	98.88
N-MNIST	ST-DFA: 2312-800-10	98.47
N-MNIST	ST-DFA-2: 2312-800-10	98.59
TI46	HM2-BP: 78-800-26	89.92
TI46	ST-DFA: 78-800-26	87.00
TI46	ST-DFA-2: 78-800-26	87.31

All SNNs are fully connected networks with a single hidden layer of 800 neurons. MNIST: 28 × 28 input resolution; N-MNIST: 2,312 input spike trains; 16-speaker TI46: 78 input spike trains.

TABLE 2 | Overheads of the fully-connected SNNs with on-chip ST-DFA-2 implemented on Xilinx ZC706 board.**MNIST (14 × 14 input resolution) @100 MHz**

	Resource utilization			Training power (mW)	Training latency (mS)	Training energy (mJ)
	LUTs	FFs	DSPs			
196-50-10	33484	6836	60	113	3.998	0.452
196-50-50-10	62989	12516	110	125	4.836	0.604
196-100-10	73027	12329	110	224	4.802	1.076
196-100-100-10	126482	23331	210	275	6.445	1.772

TI46 (16-speaker Spoken English Letters) @100 MHz

	Resource utilization			Training power (mW)	Training latency (mS)	Training energy (mJ)
	LUTs	FFs	DSPs			
78-50-26	38220	8826	76	73	3.688	0.269
78-50-50-26	74709	14641	126	87	5.123	0.445
78-100-26	64280	14096	126	113	5.089	0.575
78-100-100-26	145452	30546	226	185	7.929	1.467

As shown in **Tables 2, 3**, the implemented networks have either one or two hidden layer(s), and each hidden layer has 50 or 100 neurons. Numbers of input and output neurons are application-dependent. The training latency and training energy are for training a representative input example of the corresponding dataset using one iteration of forward and backward passes. **Table 2** indicates that the SNNs integrated with ST-DFA-2 in general have efficient FPGA resource utilization as well as low training energy dissipation.

Furthermore, with a trimmed down input size and/or constrained network size, the FPGA SNNs with on-chip ST-DFA-2 can still deliver competitive classification performance in reference to the simulated accuracies achieved at full input size and by larger networks reported in **Table 1**. For instance, the accuracy of MNIST in **Table 1** is based on full input resolution

TABLE 3 | Inference performances of the fully-connected SNNs with on-chip ST-DFA-2 measured on Xilinx ZC706 FPGA board.

MNIST (14 × 14 resolution) @100 MHz	
Accuracy (On-board) (%)	
196-50-10	94.34
196-50-50-10	94.51
196-100-10	95.72
196-100-100-10	96.27
TI46 (16-speaker English Letters) @100 MHz	
Accuracy (On-board) (%)	
78-50-26	71.63
78-50-50-26	74.95
78-100-26	75.19
78-100-100-26	84.88

TABLE 4 | Overheads of an FPGA SNN with on-chip HM2-BP vs. ST-DFA-2 (Network size:196-100-100-10).

	LUTs	FFs	DSPs	Backward phase latency (uS)
HM2-BP	154477	23462	900	17.560
ST-DFA	126482	23331	210	12.010
	Normalized LUTs (%)	Normalized FFs (%)	Normalized DSPs (%)	Normalized B-P latency (%)
HM2-BP	122	101	429	146
ST-DFA	100	100	100	100

which is 28 × 28 with a hidden layer of 800 neurons. However, we implemented on-board simulation with reduced input resolution and smaller networks due to the Xilinx ZC706 board resource limitation. We cropped each data of MNIST into 14 × 14 which causes 4X input resolution reduction and built smaller networks consists of 50 or 100 hidden neurons as shown in **Table 3**. Despite the low resolution and the small network size, SNN neural processors with on-chip ST-DFA training show competitive classification accuracy of 96.27% for MNIST, 84.88% for TI46 speech corpus, respectively.

To better illustrate the cost-effectiveness of the proposed ST-DFA algorithm, we also compare the overheads of implementing HM2-BP vs. ST-DFA-2 in a fully-connected SNN FPGA with two hidden layers in **Table 4**. Since the main difference between HM2-BP and ST-DFA is the backward pass algorithm, we designed HM2-BP in hardware based on the weight updating algorithm represented in Jin et al. (2018). Training latency of the backward pass of the corresponding SNN neural processor is also presented in the table. We do not consider forward pass latency and inference latency since they do not differ significantly in the two cases. The results in the table indicate that ST-DFA is much more efficient in terms of hardware implementation on

both resource utilization and backward pass latency compared with HM2-BP. The ST-DFA-2 based SNN neural processor saves 18% on LUTs, 76.7% on DSPs and 31.6% on backward phase latency compared with the HM2-BP based SNN.

4. DISCUSSIONS

While Direct Feedback Alignment (DFA) has been attempted, this work present first novel approach for implementing hardware spiking neural networks (SNNs) on FPGA board. This work aims to build efficient on-chip training FPGA SNN neural processor with reduced backward training latency and hardware cost while gracefully trading off classification performance. To be specific, **Table 3** shows competitive software/hardware inference accuracy despite reduce input resolution and small network size. Comparing the accuracy of ST-DFA and ST-DFA-2, performance using ST-DFA can be slightly better than using ST-DFA-2 and vice versa. This fact shows that the error may vary based on randomly initialized feedback weight matrix, which makes ST-DFA-2 still powerful. **Table 4** shows the advantages of ST-DFA-2 over HM2-BP in terms of hardware resource utilization through the DFA algorithm and the efficient design of the hardware design units. As shown in **Tables 1, 3**, this result proves the practicality of the DFA algorithm and the feasibility of implementing the ST-DFA algorithm for on-chip training of the SNN processor.

The large additional hardware overhead and backward latency of HM2-BP mainly come from the layer-by-layer error propagation and the required multiplication operations. Moreover, as the network goes deeper, the backward phase latency grows proportionally in HM2-BP, while in ST-DFA the backward latency will not affect by the network depth since the error processing is concurrently executed in all hidden layers. This property assures the scalability of ST-DFA which is promising for deeper networks. With the proposed ST-DFA algorithm, we have sidestepped the complex backpropagation and enabled cost-effective on-chip training for multi-layer SNNs.

As discussed in section 1, implementing training of SNNs using a BP algorithm suffers from high computing complexity and thus high resource utilization, while a hardware-friendly, non-BP algorithm, such as STDP, suffers from achieving good accuracies. We argue that our approach avoids high computation complexity by extending a BP-like algorithm (i.e., DFA) for SNNs while maintaining the advantage of a well-defined BP-like algorithm in terms of accuracy. To the best of our knowledge, this is the first work presenting algorithm-hardware co-optimization and demonstrating the realization of DFA, which is an efficient on-chip training algorithm, for SNNs.

For example, to compare with existing onchip works, Yin et al. (2017) presented a new BP based training algorithm for discrete-time SNNs by using a LIF neuron model with a gradient estimator. This paper introduced a ReLU-like gradient estimation method to avoid the zero-gradient issue in conventional SNNs using LIF neurons. However, as the experiment results in Yin et al. (2017) are based on off-chip training, we guess that this new BP algorithm still suffers from an efficient on-chip training

method. We think that the main difference of Yin et al. (2017) and our work is that Yin et al. (2017) proposed a new BP-based learning algorithm while our work proposes a new BP-like learning algorithm (i.e., DFA) based on a state-of-the-art BP algorithm and efficiently implement it in hardware while delivering competitive accuracy. As a small scale low-power accelerator, Zheng and Pinaki (2018) proposed a hardware-friendly STDP on-chip training algorithm. This paper focuses on capturing the estimated gradients concerning STDP behaviors. By simplifying the calculation of STDP based gradient for weight updating, this work presented an efficient on-chip learning algorithm that can be implemented on hardware. However, this paper still suffers from accuracy, and the main difference is that our work is focusing on BP-like training algorithms which have demonstrated excellent performance in recent years.

However, several challenges should be addressed to achieve more practical application. Although this paper proposes hardware-efficient designs, the resource limitation of the FPGA board does not allow large networks. For instance, we reduced the input resolution of MNIST dataset from 784 to 196 due to the limitation. While the proposed DFA based on-chip learning is demonstrated using relatively small SNNs on FPGA due to hardware resource limitations, our future work will explore a number of techniques, such as more advanced neuron model simplification, architectural level optimization, and/or a larger FPGA board to demonstrate larger-scale SNNs.

Nevertheless, the main focuses of this paper have been on extending the DFA concept proposed by Nøkland (2016) to efficient training of SNNs, and significantly reducing the hardware cost by algorithm-hardware co-optimization while maintaining a competitive accuracy. This paper proposes a novel spike-level direct feedback alignment (ST-DFA) algorithm for training multi-layer spiking neural networks (SNNs) with improved bio-plausibility and scalability over traditional backpropagation algorithms. Moreover, it is demonstrated that the ST-DFA algorithm with its hardware-friendly optimized implementation enable efficient on-chip training of FPGA SNN neural processors while delivering competitive classification performance for practical speech and image recognition tasks.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

JL implemented the on-board simulation and performed experimental studies on the Xilinx ZC706 FPGA board. JL and RZ co-designed the hardware architecture of ST-DFA including all hardware units, such as S-PSP calculation block and ST-DFA weight update block. WZ and PL developed the theoretical approach of Direct Feedback Alignment and provided software simulation. PL defined and directed the overall research. JL, RZ, WZ, YL, and PL wrote the paper. RZ and YL performed this work while being at Texas A&M University.

FUNDING

This material is based upon work supported by the National Science Foundation (NSF) under Grant Nos. 1639995 and 1948201, and the Semiconductor Research Corporation (SRC) under task #2692.001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the

authors and do not necessarily reflect the views of NSF, SRC, University of California Santa Barbara, Texas A&M University and their contractors. The authors declare that this study received funding from the Semiconductor Research Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

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
- 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
- Chankyu, L., Syed Shakib, S., and Kaushik, R. (2019). Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *arXiv [Preprint]*. arXiv:1903.06379.
- Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). EMNIST: an extension of mnist to handwritten letters. *arXiv [Preprint]*. arXiv:1702.05373 doi: 10.1109/IJCNN.2017.7966217
- 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
- Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.
- 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*, pages 7005–7015.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324.
- 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
- Liberman, M., Amsler, R., Church, K., Fox, E., Hafner, C., Klavans, J., et al. (1991). TI 46-word LDC93S9.
- 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
- Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* 7:13276. doi: 10.1038/ncomms13276
- Lyon, R. (1982). “A computational model of filtering, detection, and compression in the cochlea,” in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP’82*, Vol. 7 (Paris: IEEE), 1282–1285.
- Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671.
- 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
- Neftci, E. O., Augustine, C., Paul, S., and Deterakis, G. (2017). Event-driven random back-propagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324
- Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). “Phased lstm: accelerating recurrent network training for long or event-based sequences,” in *Advances in Neural Information Processing Systems*, eds D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc), 3882–3890.
- Nøkland, A. (2016). “Direct feedback alignment provides learning in deep neural networks,” in *Advances in Neural Information Processing Systems*, eds D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc), 1037–1045.
- 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
- Panda, P., Aparna, A., and Kaushik, R. (2019). Towards scalable, efficient and accurate deep spiking neural networks with backward residual connections, stochastic softmax and hybridization. *arXiv [Preprint]*. arXiv:1910.13931.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988a). Learning representations by back-propagating errors. *Cogn. Model.* 5:1.
- Rumelhart, D. E., McClelland, J. L., and Group, P. R. (1988b). *Parallel Distributed Processing*, Vol. 1. Cambridge, MA: MIT Press.
- Schrauwen, B., and Van Campenhout, J. (2003). “BSA, a fast and accurate spike train encoding scheme,” in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 4 (Portland, OR: IEEE), 2825–2830.
- Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560.
- 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
- Yin, S., Venkataramanaiah, S. K., Chen, G. K., Krishnamurthy, R., Cao, Y., Chakrabarti, C., et al. (2017). “Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations,” in *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, (Turin), 1–5.
- Zheng, N., and Pinaki, M. (2018). “A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks,” in *International Symposium on Circuits and Systems (ISCAS)*, (Florence), 1–5.

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.

Copyright © 2020 Lee, Zhang, Zhang, Liu and Li. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition

Jibin Wu^{1*}, Emre Yilmaz¹, Malu Zhang¹, Haizhou Li^{1,2} and Kay Chen Tan³

¹ Department of Electrical and Computer Engineering, National University of Singapore, Singapore, Singapore, ² Faculty for Computer Science and Mathematics, University of Bremen, Bremen, Germany, ³ Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong

Artificial neural networks (ANN) have become the mainstream acoustic modeling technique for large vocabulary automatic speech recognition (ASR). A conventional ANN features a multi-layer architecture that requires massive amounts of computation. The brain-inspired spiking neural networks (SNN) closely mimic the biological neural networks and can operate on low-power neuromorphic hardware with spike-based computation. Motivated by their unprecedented energy-efficiency and rapid information processing capability, we explore the use of SNNs for speech recognition. In this work, we use SNNs for acoustic modeling and evaluate their performance on several large vocabulary recognition scenarios. The experimental results demonstrate competitive ASR accuracies to their ANN counterparts, while require only 10 algorithmic time steps and as low as 0.68 times total synaptic operations to classify each audio frame. Integrating the algorithmic power of deep SNNs with energy-efficient neuromorphic hardware, therefore, offer an attractive solution for ASR applications running locally on mobile and embedded devices.

Keywords: deep spiking neural networks, automatic speech recognition, tandem learning, neuromorphic computing, acoustic modeling

OPEN ACCESS

Edited by:

Huajin Tang,
Zhejiang University, China

Reviewed by:

Federico Corradi,
Imec, Netherlands
Juan Pedro Dominguez-Morales,
University of Seville, Spain

*Correspondence:

Jibin Wu
jibin.wu@u.nus.edu

Specialty section:

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

Received: 19 November 2019

Accepted: 24 February 2020

Published: 17 March 2020

Citation:

Wu J, Yilmaz E, Zhang M, Li H and
Tan KC (2020) Deep Spiking Neural
Networks for Large Vocabulary
Automatic Speech Recognition.
Front. Neurosci. 14:199.
doi: 10.3389/fnins.2020.00199

1. INTRODUCTION

Automatic speech recognition (ASR) has enabled the voice interface of mobile devices and smart home appliances in our everyday life. The rapid progress in the integration of voice interfaces has been viable on account of the remarkable performance of the ASR systems using artificial neural networks (ANN) for acoustic modeling (Lippmann, 1989; Lang et al., 1990; Hinton et al., 2012; Yu and Deng, 2015). Various ANN architectures, either feedforward or recurrent, have been investigated for modeling the acoustic information preserved in speech signals (Dahl et al., 2012; Graves et al., 2013; Abdel-Hamid et al., 2014).

The performance gains come with immense computational requirements often due to the time-synchronous processing of input audio signals. Several techniques have been proposed to reduce the computational load and memory storage of ANNs by reducing the number of parameters that have to be used for inference (Sainath et al., 2013; Xue et al., 2013; He et al., 2014; Povey et al., 2018). Another common solution, for reducing the processing load, uses a wake word or phrase to control the access to speech recognition services (Zehetner et al., 2014; Sainath and Parada, 2015; Wu M. et al., 2018). Moreover, most devices with voice control rely on cloud-based ASR engines rather than local on-device solutions. The necessity of online processing of speech via cloud computing comes with various concerns, such as data security and processing speed, etc. There have been multiple

efforts to develop on-device ASR solutions in which the speech signal is processed locally using the computational resources of mobile devices (Lei et al., 2013; McGraw et al., 2016).

Alternatively, event-driven models such as spiking neural networks (SNNs) inspired by the human brain have attracted ever-growing attention in recent years. The human brain is remarkably efficient and capable of performing complex perceptual and cognitive tasks. Notably, the adult's brain only consumes about 20 watts to solve complex tasks that are equivalent to the power consumption of a dim light bulb (Laughlin and Sejnowski, 2003). While brain-inspired ANNs have demonstrated great capabilities in many perceptual (He et al., 2016; Xiong et al., 2017) and cognitive tasks (Silver et al., 2017), these models are computationally intensive and memory inefficient to operate as compared to the biological brains. Unlike ANNs, asynchronous and event-driven information processing of SNNs resembles the computing paradigm that observed in the human brains, whereby the energy consumption matches the activity levels of sensory stimuli. Given temporally sparse information transmitted in the surrounding environment, the event-driven computation, therefore, exhibits great computational efficiency than the synchronous computation used in ANNs.

Neuromorphic computing (NC), as a non-von Neumann computing paradigm, mimics the event-driven computation of the biological neural systems with SNN in silicon. The emerging neuromorphic computing architectures (Furber et al., 2012; Merolla et al., 2014; Davies et al., 2018) leverage on the massively parallel, low-power computing units to support spike-based information processing. Notably, the design of co-located memory and computing units effectively circumvents the von Neumann bottleneck of low-bandwidth between memory and the processing units (Monroe, 2014). Therefore, integrating the algorithmic power of deep SNNs with the compelling energy efficiency of NC hardware represents an intriguing solution for pervasive machine learning tasks and always-on applications. Furthermore, growing research efforts are devoted to developing novel non-volatile memory devices for ultra-low-power implementation of biological synapses and neurons (Tang et al., 2019).

Some preliminary work on SNN-based phone classification or small-vocabulary speech recognition systems have been explored in Jim-Shih Liaw and Berger (1998), Näger et al. (2002), Loisel et al. (2005), Holmberg et al. (2005), Kröger et al. (2009), Tavanai and Maida (2017a,b), Wu et al. (2018a), Wu et al. (2018b), Zhang et al. (2015), Zhang et al. (2019), Bellec et al. (2018), Wu et al. (2019b), and Pan et al. (2018). However, these SNN-based ASR systems are far from the scale and complexity of modern commercialized ANN-based ASR systems. It is mainly due to lacking effective training algorithms for deep SNNs and efficient software toolbox for SNN-based ASR systems.

Due to the discrete and non-differentiable nature of spike generation, the powerful error back-propagation algorithm is not directly applicable to the training of deep SNNs. Recently, considerable research efforts are devoted to addressing this problem and the resulting learning rules can be broadly categorized into the SNN-to-ANN conversion (Cao et al., 2015; Diehl et al., 2015), back-propagation through time with surrogate

gradient (Wu Y. et al., 2018; Neftci et al., 2019; Wu et al., 2019a) and tandem learning (Wu et al., 2019c). Despite several successful attempts on the large-scale image classification tasks with deep SNNs (Rueckauer et al., 2017; Hu et al., 2018; Sengupta et al., 2019; Wu et al., 2019c), their applications to the large-vocabulary continuous ASR (LVCSR) tasks remain unexplored. In this work, we explore an SNN-based acoustic model for LVCSR using a recently proposed tandem learning rule (Wu et al., 2019c) that supports an efficient and rapid inference.

To summarize, the main contributions of this work are threefold:

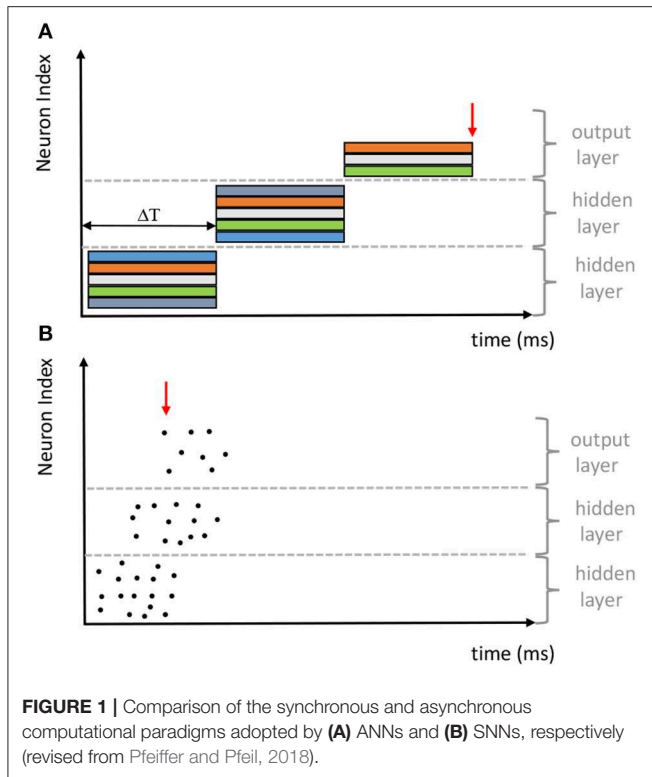
- **Large-Vocabulary Automatic Speech Recognition with SNNs.** We explored the SNN-based acoustic models for large-vocabulary automatic speech recognition tasks. The SNN-based ASR systems achieved competitive accuracy on par with their ANN counterparts across the phone recognition, low-resourced ASR and large-vocabulary ASR tasks. To the best of our knowledge, this is the first work that successfully applied SNNs to the LVCSR task.
- **Toward Rapid and Energy-Efficient Speech Recognition.** Our preliminary study of an SNN-based acoustic model has revealed compelling prospect of rapid inference and unprecedented energy efficiency of a neuromorphic approach. Specifically, SNNs can classify each audio frame accurately with only 10 algorithmic time steps while require as low as 0.68 times total synaptic operations to their ANN counterparts.
- **SNN-Based ASR Toolkit.** We demonstrate that SNN-based acoustic models can be effectively developed in PyTorch and easily integrated into the PyTorch-Kaldi Speech Recognition Toolkit (Ravanelli et al., 2019) for rapid development of SNN-based ASR systems.

The rest of the paper is organized as follows: In section 2, we first give an overview of spiking neural networks, large vocabulary ASR systems, and existing SNN-based ASR systems. In section 3, we introduce the spiking neuron model and the neural coding scheme that converts acoustic features into spike-based representation. We further present a recently introduced tandem learning framework for SNN training and how it is used to train deep SNN-based acoustic models. In section 4, we present experimental results on the learning capability and energy efficiency of SNN-based acoustic models across three different types of recognition tasks including phone recognition, low-resourced and standard large-vocabulary ASR, and compare those to the ANN-based implementations. Finally, a discussion on the experimental findings is given in section 5.

2. FUNDAMENTALS AND RELATED WORK

2.1. Spiking Neural Networks

The third generation spiking neural networks are originally studied as models to describe the information processing in the biological neural networks, wherein the information is communicated and exchanged via stereotypical action potentials or spikes (Gerstner and Kistler, 2002). Neuroscience studies reveal that the temporal structure and frequency of these spike

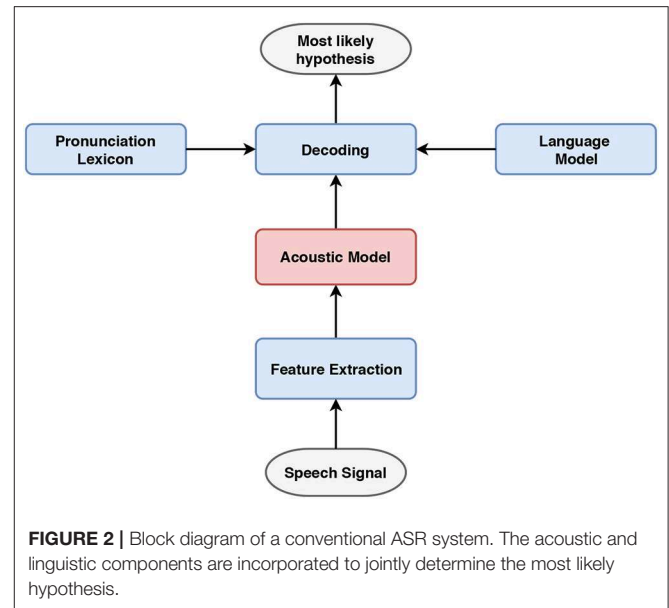


trains are both important information carriers in the biological neural networks. As will be introduced in section 3.1, the spiking neuron operates asynchronously and integrates the synaptic current from its incoming spike trains. An output spike is generated from the spiking neuron whenever its membrane potential crosses the firing threshold, and this output spike will be propagated to the connected neurons via the axon.

Motivated by the same connectionism principle, SNNs share the same network architectures, either feedforward or recurrent, with the conventional ANNs that use analog neurons. As shown in **Figure 1**, the early classification decision can be made from the SNN since the generation of the first output spike. However, the quality of the classification decision is typically improved over time with more evidence accumulated. It differs significantly from the synchronous information processing of the conventional ANNs, where the output layer needs to wait until all preceding layers are fully updated. Therefore, despite information is transmitted and processed at a speed that is several orders of magnitude slower in neural substrates than signal processing in modern transistors, biological neural systems can perform complex tasks rapidly. For more overviews about SNNs and their applications, we refer readers to Pfeiffer and Pfeil (2018) and Tavanaei et al. (2019).

2.2. Large Vocabulary Automatic Speech Recognition

As shown in **Figure 2**, conventional ASR systems use acoustic and linguistic information preserved in three distinct components to convert speech signals to the corresponding text: (1) an acoustic model for preserving the statistical



representations of different speech units, e.g., phones, from speech features, (2) a language model for assigning probabilities to the co-occurring word sequences and (3) a pronunciation lexicon for mapping the phonetic transcriptions to orthography. These resources are jointly used to determine the most likely hypothesis in the decoding stage.

Acoustic modeling can be achieved by using various statistical models such as Gaussian Mixture Models (GMM) for assigning frame-level phone posteriors in conjunction with a Hidden Markov Model (HMM) for duration modeling (Yu and Deng, 2015). More recently, ANN-based approaches have become the standard acoustic models providing state-of-the-art performance across a wide spectrum of ASR tasks (Hinton et al., 2012). Together with numerous ANN architectures explored for acoustic modeling, several end-to-end ANN architectures have been proposed for directly mapping speech features to text with optional use of the other linguistic components (Graves and Jaitly, 2014; Chan et al., 2016; Watanabe et al., 2017).

The probabilistic definition of acoustic modeling becomes more evident via the Bayesian formulation of the speech recognition task. Given a target speech signal that segmented into T overlapped frames, the resulting frame-wise features can be represented as $\mathbf{O} = [\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T]$. An ASR system assigns the probability $P(\mathbf{W}|\mathbf{O})$ to all possible word sequences $\mathbf{W} = [w_1, w_2, \dots]$, and the word sequence $\hat{\mathbf{W}}$ with the highest probability is the recognized output,

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \quad (1)$$

The probability $P(\mathbf{W}|\mathbf{O})$ can be decomposed into two parts by applying the Bayes' rule as below,

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} \quad (2)$$

$P(\mathbf{O})$ can be omitted as it does not depend on \mathbf{W} . This results in

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \quad (3)$$

which formally defines the theoretical foundation of the conventional ASR systems. $P(\mathbf{W})$ is the prior probability of the word sequence \mathbf{W} and this probability is provided by the language model which is trained on a large written corpus of the target language. $P(\mathbf{O}|\mathbf{W})$ is the *likelihood* of the observed feature sequence \mathbf{O} given the word sequence \mathbf{W} , and this probability is associated with the acoustic model. The acoustic model captures the information about the acoustic component of speech signals, aiming to classify different acoustic units accurately. Traditionally, each phone in the phonetic alphabet is modeled using multiple three-state HMM models for different preceding and following phonetic context (triphone) (Lee, 1990). The emission probability of these HMM states are shared (tied) among different models to reduce the number of model parameters (Hwang and Huang, 1993). The output layer of the ANN-based acoustic model is designed accordingly and trained to assign these frame-level tied triphone HMM state (senone) probabilities (Dahl et al., 2012). The output layer uses the softmax function to normalize the output into a probability distribution. These values are scaled with the prior probabilities of each class, obtained from the training data, to determine the likelihood values. These likelihood values are later combined with the probabilities assigned by the language model during the decoding stage so as to find the most likely hypothesis.

Speech features, used as the inputs to the acoustic model, describe the spectrotemporal dynamics of the speech signal and discriminate among different phones in the target language. Mel-frequency cepstral coefficients (MFCC) (Davis and Mermelstein, 1980) features are commonly used in conjunction with the GMM-HMM acoustic model. The MFCC features are extracted by (1) performing short-time Fourier transform, (2) applying triangular Mel-scaled filter banks to calculate the power at each Mel frequency in log domain (FBANK) and (3) performing a discrete cosine transform to decorrelate the FBANK features. The third step is often skipped and FBANK features are often used when training ANN-based acoustic models since these models can handle correlation among features. In this work, we incorporate deep SNNs for acoustic modeling instead of the conventional ANNs and compare their ASR performance in different ASR scenarios including phone recognition, low-resourced and standard large vocabulary ASR. The ASR performance obtained using popular speech features have been reported to explore the impact of the feature representation space and its dimensionality for SNN-based acoustic models.

2.3. Speech Recognition With Spiking Neural Network

SNNs are well-suited for representing and processing spatial-temporal signals, they hence possess great potentials for speech recognition tasks. Tavanaei and Maida (2017a,b) proposed SNN-based feature extractors to extract discriminative features

from the raw speech signal using unsupervised spiking-timing-dependent plasticity (STDP) rule. While connecting these SNN-based feature extractors with Support Vector Machine (SVM) or Hidden Markov Model (HMM) classifiers, competitive classification accuracies were demonstrated on the isolated spoken digit recognition task. Wu et al. (2018a,b) introduced a SOM-SNN framework for environmental sound and speech recognition. In this framework, the biological-inspired self-organizing map (SOM) is utilized for feature representation, which maps frame-based acoustic features into a spike-based representation that is both sparse and discriminative. The temporal dynamic of the speech signal is further handled by the SNN classifier. Zhang et al. (2019) presented a fully SNN-based speech recognition framework, wherein the spectral information of consecutive frames are encoded with threshold coding and subsequently classified by the SNN that is trained with a novel membrane potential-driven aggregate-labeling learning algorithm.

Recurrent network of spiking neurons (RSNNs) exhibit greater memory capacity than the aforementioned feedforward frameworks. They can capture long temporal information that are useful for speech recognition tasks. In Zhang et al. (2015), Zhang et al. presented a spiking liquid-state machine (LSM) speech recognition framework which is attractive for low-power very-large-scale-integration (VLSI) implementation. Bellec et al. recently demonstrated state-of-the-art phone recognition accuracy on the TIMIT dataset by adding neuronal adaptation mechanism to the vanilla RSNNs (Bellec et al., 2018). It is the first time that RSNNs approaching the performance of LSTM networks (Greff et al., 2016) on the speech recognition task. These preliminary works on the SNN-based ASR systems are however limited to the phone classification or small vocabulary isolated spoken digit recognition tasks. In this work, we apply deep SNNs to LVCSR tasks and demonstrate competitive accuracies over the ANN-based ASR systems.

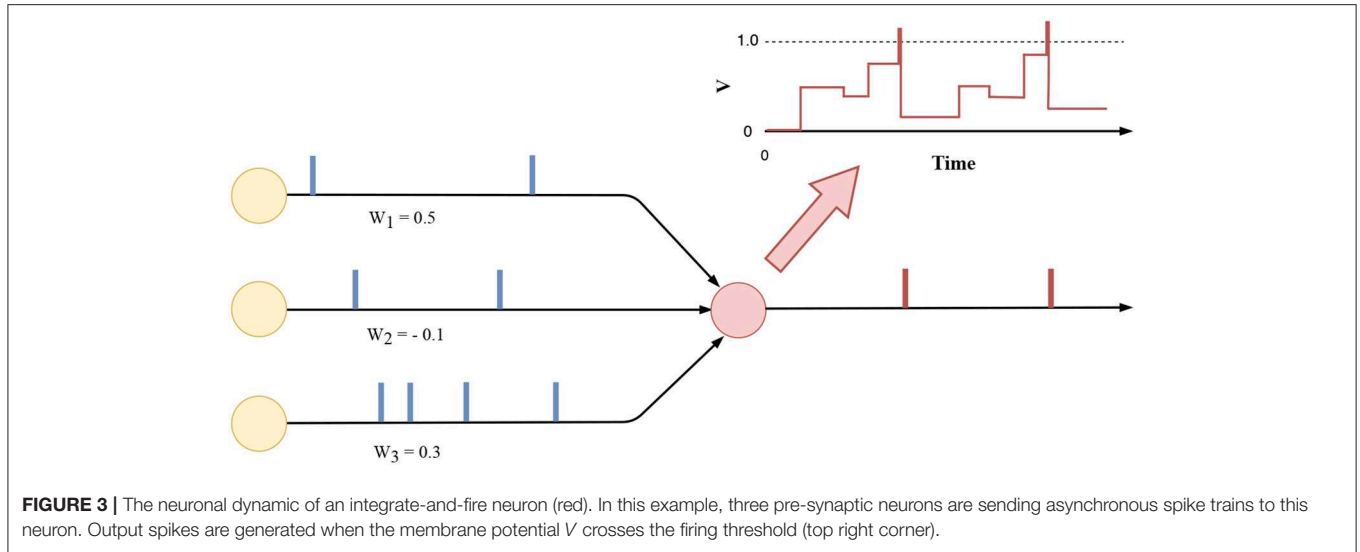
3. METHODS

3.1. Spiking Neuron Model

As shown in **Figure 4**, the frame-based features are first extracted and input into the SNN-based acoustic models. Given the short temporal duration of segmented frames and the slow variation of speech signals, these features are typically assumed to be stationary over the short time-period of segmented frames. In this work, we use the integrate-and-fire (IF) neuron model with reset by subtraction scheme (Rueckauer et al., 2017), which can effectively process these stationary frame-based features with minimal computational costs. Although IF neurons do not emulate rich temporal dynamics of biological neurons, they are however ideal for working with the neural representation that employed in this work, where spike timings play an insignificant role.

At each time step t of a discrete-time simulation, with a total number of time steps N_s , the incoming spikes to neuron j at layer l are transduced into synaptic current as follows

$$z_j^l(t) = \sum_i w_{ji}^{l-1} \cdot \theta_i^{l-1}(t) + b_j^l \quad (4)$$



where $\theta_i^{l-1}(t)$ indicates the occurrence of an input spike from afferent neuron i at time step t . In addition, the w_{ji}^{l-1} denotes the synaptic weight that connects presynaptic neuron i from layer $l-1$. Here, b_j^l can be interpreted as a constant injecting current. As shown in **Figure 3**, neuron j integrates the input current $z_j^l(t)$ into its membrane potential $V_j^l(t)$ as per Equation (5). The $V_j^l(0)$ is reset and initialized to zero for every new frame-based feature input. Without loss of generality, a unitary membrane resistance is assumed here. An output spike is generated whenever $V_j^l(t)$ crosses the firing threshold ϑ (Equation 6), which we set to a value of 1 for all the experiments by assuming that all synaptic weights are normalized with respect to the ϑ .

$$V_j^l(t) = V_j^l(t-1) + z_j^l(t) - \vartheta \cdot \theta_j^l(t-1) \quad (5)$$

$$\theta_j^l(t) = \Theta(V_j^l(t) - \vartheta) \text{ with } \Theta(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

According to Equations (4) and (5), the free aggregated membrane potential of neuron j (no firing) in layer l can be expressed as

$$V_j^{lf} = \sum_i w_{ji}^{l-1} \cdot c_i^{l-1} + b_j^l \cdot N_s \quad (7)$$

where c_i^{l-1} is the input spike count from pre-synaptic neuron i at layer $l-1$ as per Equation (8).

$$c_i^{l-1} = \sum_{t=1}^{N_s} \theta_i^{l-1}(t). \quad (8)$$

The V_j^{lf} summarizes the aggregate membrane potential contributions of the incoming spikes from pre-synaptic neurons while ignoring their temporal structures. As will be explained in the tandem learning framework section, this intermediate quantity links the SNN layers to the coupled ANN layers for parameter optimization.

3.2. Neural Coding Scheme

SNNs process information transmitted via spike trains, therefore, special mechanisms are required to encode the continuous-valued feature vectors into spike trains and decode the classification results from the activity of output neurons. To this end, we adopt the spiking neural encoding scheme that proposed in Wu et al. (2019c). This encoding scheme first transforms frame-based input feature vector X^0 (e.g., MFCC or FBANK features), where $X^0 = [x_1^0, x_2^0, \dots, x_n^0]^T$, through a weighted layer of rectified linear unit (ReLU) neurons as follows

$$V_j^{0f}(0) \equiv a_j^0 = \rho(\sum_i w_{ji}^0 \cdot x_i^0 + b_j^0) \quad (9)$$

where w_{ji}^0 is the strength of the synaptic connection between the input x_i^0 and ReLU neuron j . The b_j^0 is the corresponding bias term of the neuron j , and $\rho(\cdot)$ denotes the ReLU activation function. The free aggregate membrane potential $V_j^{0f}(0)$ is defined to be equal to the activation value a_j^0 of the ReLU neuron j . We distribute this quantity over the encoding time window N_s and represent it via spike trains as per Equations (10) and (11).

$$\theta_j^0(t) = \Theta(V_j^{0f}(t-1) - \vartheta) \quad (10)$$

$$V_j^{0f}(t) = V_j^{0f}(t-1) - \vartheta \cdot \theta_j^0(t) \quad (11)$$

Altogether, the spike train s^0 and spike count c^0 that output from the neural encoding layer can be represented as follows

$$s^0 = \{\theta^0(1), \dots, \theta^0(N_s)\} \quad (12)$$

$$c^0 = \sum_{t=1}^{N_s} \theta^0(t) \quad (13)$$

This encoding layer performs weighted transformation inside an end-to-end learning framework. It transforms the original

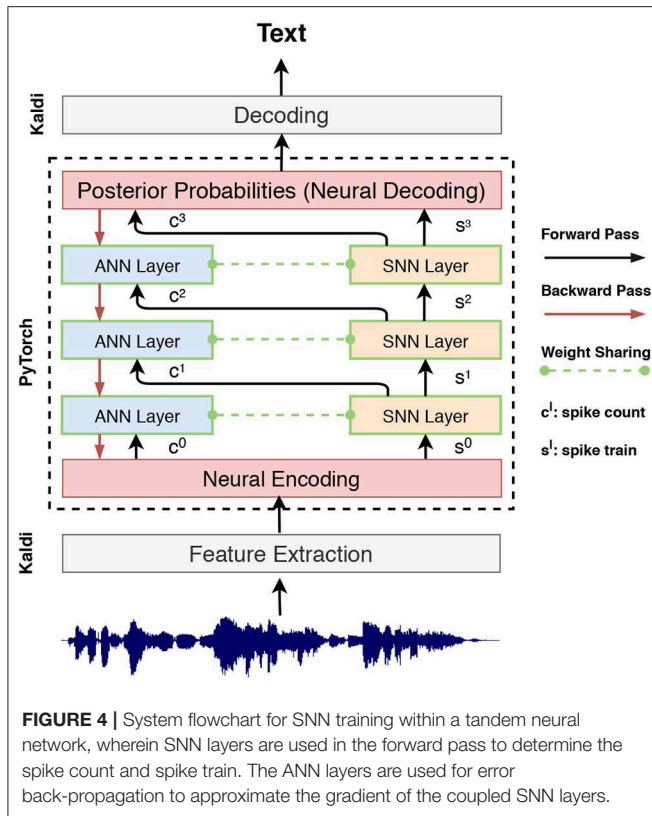


FIGURE 4 | System flowchart for SNN training within a tandem neural network, wherein SNN layers are used in the forward pass to determine the spike count and spike train. The ANN layers are used for error back-propagation to approximate the gradient of the coupled SNN layers.

input representation to match the size of the encoding time window N_s and represents the transformed information via spike counts. This encoding scheme is beneficial for rapid inference since the input information can be effectively encoded within a short encoding window. Start from this neural encoding layer, as shown in **Figure 4**, we input the spike count c^l and s^l to subsequent ANN and SNN layers for tandem learning.

To ensure smooth learning with high precision error gradients derived at the output layer, we use the free aggregate membrane potential of output spiking neurons for neural decoding. Considering that the dimensionality of input feature vectors and output classes are much smaller than that of hidden layers, the computation required will be limited when deploying these two layers onto the edge devices.

3.3. Tandem Learning for Training Deep SNNs

Here, we present a recently proposed SNN learning rule, under the tandem neural network configuration, that exploits a connection between the activation value of ANN neurons and the spike count of IF neurons. As the input features are effectively encoded as spike counts, the temporal structure of the spike trains carries negligible information. The effective non-linear transformation of SNN layers therefore can be summarized as

$$c_j^l = f(s^{l-1}; w_j^{l-1}, b_j^l) \quad (14)$$

where $f()$ denotes the transformation performed by spiking neurons. However, due to the state-dependent nature of spike generation, it is not viable to determine an analytical expression from s^{l-1} to c_j^l directly. Therefore, we simplify the spike generation process by assuming the resulting synaptic currents from s^{l-1} are evenly distributed over the encoding time window. As such, the interspike interval can be determined as follows

$$ISI_j^l = \rho \left(\frac{\vartheta}{V_j^{lf}/N_s} \right) = \rho \left(\frac{\vartheta}{(\sum_i w_{ji}^{l-1} c_i^{l-1} + b_j^l \cdot N_s)/N_s} \right) \quad (15)$$

Hence, the approximated “spike count” a_j^l can be derived according to

$$a_j^l = \frac{N_s}{ISI_j^l} = \frac{1}{\vartheta} \cdot \rho \left(\sum_i w_{ji}^{l-1} c_i^{l-1} + b_j^l \cdot N_s \right) \quad (16)$$

Given a unitary firing threshold ϑ , a_j^l can be effectively determined from an ANN layer of ReLU neurons by setting the spike count c_i^{l-1} as the input and the aggregated constant injecting current $b_j^l \cdot N_s$ as the bias term. This simplification of spike generation process allows the spike-train level error gradients to be approximated from the ANN layer. Wu et al. (2019c) have revealed that the cosine distances between the approximated ‘spike count’ a^l and the actual SNN output spike count c^l are exceedingly small in a high dimensional space, suggesting high quality error gradients can be approximated from the coupled ANN layers.

Based on this formulation, we constructed tandem neural networks as shown in **Figure 4**. During the activation forward propagation, the SNN layers are used to determine the exact spike representation which then propagate the aggregate spike counts and spike trains to the subsequent ANN and SNN layers, respectively. This interlaced layer structure ensures the information that forward propagated to the coupled ANN and SNN layers are synchronized. It worth noting that the ANN is just an auxiliary structure to facilitates the training of SNN, while only SNN is used during inference. The details of this tandem learning rule are provided in the Algorithm 1.

3.4. SNN-Based Acoustic Modeling

To train the deep SNN-based acoustic models, which is the main contribution of this work, several popular speech features have been extracted from the training recordings as described in section 2.2. Before being fed into the SNNs, these input speech features are contextualized by splicing multiple frames so as to exploit more temporal context information. Before training the SNN-based acoustic model, alignments of the speech features with the target senone labels are obtained using a conventional GMM-HMM-based ASR system similar to that described in Dahl et al. (2012). These frame-level alignments enable the training of the deep SNN acoustic model with the tandem learning approach. During the training, the deep SNN learns to map input speech features to posterior probabilities of senones (cf.

Algorithm 1: Pseudo Codes For The Tandem Learning Rule

Input: Input frame-based feature vectors X^0 , target label Y , network parameters w , neural encoding window size N_s

Output: Updated network parameters w

Forward Pass:

$c^0, s^0 = \text{Neural_Encoding}(X^0)$

for layer $l = 1$ to $N-1$ **do**

 // State Update of the ANN Layer

$a^l = \text{ANN.layer}[l].\text{forward}(c^{l-1}, w^{l-1}) *$

for $t = 1$ to N_s **do**

 // State Update of the SNN Layer

$s^l[t] = \text{SNN.layer}[l].\text{forward}(s^{l-1}[t], w^{l-1})$

 // Update the Spike Count

$c^l = \sum_{t=1}^{N_s} s^l[t]$

 /* Neural Decoding with the Aggregate Membrane Potential */

$\text{output} = \text{ANN.layer}[N].\text{forward}(c^{N-1}, w^{N-1})$

Loss: $E = \text{LossFunction}(Y, \text{output})$

Backward Pass:

$\frac{\partial E}{\partial a^N} = \text{LossGradient}(Y, \text{output})$

for layer $l = N-1$ to 1 **do**

 // Gradient Update through the ANN Layer

$\frac{\partial E}{\partial a^{l-1}}, \frac{\partial E}{\partial w^{l-1}} = \text{ANN.layer}[l].\text{backward}(\frac{\partial E}{\partial a^l}, c^{l-1}, w^{l-1})$

Update parameters of the ANN layer based on the calculated gradients.

Copy the updated parameters to the corresponding SNN layer.

Note:

* For inference, state updates are performed on the SNN layers entirely.

in this work, we refer the reader to Povey et al. (2012). In the following sections, we describe the ASR experiments conducted to evaluate the recognition performance of the proposed SNN-based acoustic modeling in several recognition scenarios.

3.5. Training and Evaluation

3.5.1. Datasets

The performance of the proposed SNN-based acoustic models is investigated in three different ASR tasks: (1) phone recognition using the TIMIT corpus (Garofolo et al., 1993), (2) low-resourced ASR task using the FAME code-switching Frisian-Dutch corpus (Yilmaz et al., 2016a) and (3) standard large-vocabulary continuous ASR task using the Librispeech corpus (Panayotov et al., 2015). All speech data used in the experiments has a sampling frequency of 16 kHz.

The train, development and test sets of the standard TIMIT corpus contain 3,696, 400, and 192 utterances from 462, 50, and 24 speakers, respectively. Each utterance is phonetically transcribed using a phonetic alphabet consisting of 48 phones in total. The training data of the FAME corpus comprises of 8.5 and 3 h of broadcast speech from Frisian and Dutch speakers, respectively. The training utterances are spoken by 382 speakers in total. This bilingual dataset contains Frisian-only and Dutch-only utterances as well as mixed utterances with inter-sentential, intra-sentential and intra-word code-switching (Myers-Scotton, 1989). The development and test sets consist of 1 h of speech from Frisian speakers and 20 min of speech from Dutch speakers each. The total number of speakers is 61 in the development set and 54 in the test set.

The Librispeech corpus contains 1,000 h of reading speech in total collected from audiobooks. This publicly available corpus¹ has been considered as a popular benchmark for ASR algorithms with multiple training and testing settings. In the ASR experiments, we train acoustic models using the 100 (train_clean_100) and 360 (train_clean_360) h of speech and apply these models to the clean development (dev_clean) and test (test_clean) sets. Further details about this corpus can be found in Panayotov et al. (2015).

3.5.2. Implementation Details

All ASR experiments are performed using the PyTorch-Kaldi ASR toolkit (Ravanelli et al., 2019). This recently introduced toolkit inherits the flexibility of PyTorch toolkit (Paszke et al., 2019) for ANN-based acoustic model development and the efficiency of Kaldi ASR toolkit (Povey et al., 2011). We implement the SNN tandem learning rule in PyTorch and integrate it into the PyTorch-Kaldi toolkit for training the proposed SNN-based acoustic models (cf. **Figure 4**). The PyTorch implementation of the described SNN acoustic models is public available online². For the baseline ANN models, the standard multi-layer perceptron recipes are used. The Kaldi toolkit is used for obtaining the initial alignments, feature extraction, graph creation, and decoding.

¹www.openslr.org/resources/12

²https://github.com/deepsike/snn-for-asr

section 2.2) by passing the input speech frames through multiple layers of spiking neurons.

During the inference phase, the acoustic scores provided by the trained SNN model are combined with the information stored in the language model and pronunciation lexicon. It is a common practice to use the weighted finite state transducers (WFST) (Mohri et al., 2002) as a unified representation of different ASR resources for creating the search graph containing possible hypotheses. The main motivation for using the WFST-based decoding is: (1) the straightforward composition of different ASR resources for constructing a mapping from HMM states to word sequences and (2) the existence of efficient search algorithms operating on WFST that speed up the decoding process. As a result of the search process, the most likely hypotheses are found and stored in the form of a lattice. The ASR output is chosen based on the weighted sum of the acoustic and language model scores belonging to hypotheses in the lattice. For further details of the WFST-based decoding approach used

For all recognition scenarios, ANNs and SNNs are constructed with 4 hidden layers and 2,048 hidden units each using the ReLU activation function. Each fully-connected layer is followed by a batch normalization layer and a dropout layer with a drop probability of 10% to prevent overfitting. We train these models using various popular speech features including the 13-dimensional Mel-frequency cepstral coefficient (MFCC) feature, 23-dimensional Mel-filterbank (FBANK) feature, and higher resolution 40-dimensional MFCC and FBANK features. We further extract feature space maximum likelihood linear regression (FMLLR) (Gales, 1998) features to explore the impact of speaker-dependent features. All features include the deltas and delta-deltas; mean and variance normalization are applied before the splicing. The time context size is set to 11 frames by concatenating 5 frames preceding and following. All features are encoded within a short time window of 10-time steps for SNN simulations.

The neural network training is performed by mini-batch Stochastic Gradient Descent (SGD) with an initial learning rate of 0.08 and a minibatch size of 128. The learning rate is halved if the improvement is less than a preset threshold of 0.001. The final acoustic models of the TIMIT and FAME corpora are obtained after 24 training epochs, while the models of the Librispeech corpus are trained for 12 epochs.

For the TIMIT and Librispeech ASR tasks, we follow the same language model (LM) and pronunciation lexicon preparation pipeline as provided in the corresponding Kaldi recipes³. The smallest 3-gram LM (tgsma11) of the Librispeech corpus is used to create the graph for the decoding stage. The details of the LM and lexicon used in the FAME recognition task are given in Yilmaz et al. (2018).

3.5.3. Evaluation Metrics

3.5.3.1. ASR performance

The phone recognition on the TIMIT corpus is reported in terms of the phone error rate (PER). The word recognition accuracies on the FAME and Librispeech corpora are reported in terms of word error rate (WER). Both metrics are calculated as the ratio of all recognition errors (insertion, deletion, and substitution) and the total number of phones or words in the reference transcriptions.

3.5.3.2. Energy efficiency: counting synaptic operations

To compare the energy efficiency of ANN and its equivalent SNN implementation, we follow the convention from NC community and compute the total synaptic operations *SynOps* that required to perform a certain task (Merolla et al., 2014; Rueckauer et al., 2017; Sengupta et al., 2019). For ANN, the total synaptic operations [Multiply-and-Accumulate (MAC)] per classification is defined as follows

$$SynOps = \sum_{l=1}^L f_{in}^l \cdot N_l \quad (17)$$

where f_{in}^l denotes the number of fan-in connections to each neuron in layer l , and N_l refers to the number of neurons in layer l . In addition, L denotes the total number of network layers. Hence, given a particular network configuration, the total synaptic operations required per classification is a constant number that jointly determined by f_{in}^l and N_l .

While for SNN, as per Equation (18), the total synaptic operations (Accumulate (AC)) required per classification are correlated with the spiking neurons' firing rate, the number of fan-out connections f_{out} to neurons in the subsequent layer as well as the simulation time window N_s .

$$SynOps = \sum_{t=1}^{N_s} \sum_{l=1}^{L-1} \sum_{j=1}^{N_l} f_{out,j}^l \cdot s_j^l(t) \quad (18)$$

where $s_j^l(t)$ indicates whether a spike is generated by neuron j of layer l at time instant t .

4. RESULTS

4.1. Phone Recognition on TIMIT Corpus

We report the PER on the development and test sets of TIMIT corpus in **Table 1**, with numbers in bold being the best performance given by the speaker-independent features. ASR performances of other state-of-the-art systems using various ANN and SNN architectures are given in the upper panel for reference purposes. As the results shown in **Table 1**, the proposed SNN-based acoustic models are applicable to different speech features and provide comparable or slightly worse ASR performance than the ANNs with the same network structure. In particular, the ANN system trained with the standard 13-dimensional FBANK feature achieves the best PER of 16.9% (18.5%) on the development (test) set. The equivalent SNN system using the same feature achieves slightly worse PER of 17.3% (18.7%) on the development (test) set. Although the state-of-the-art ASR systems (Ravanelli et al., 2018) give approximately 1% lower PER than the proposed SNN-based phone recognition system, it is largely credit to the longer time context explored by the recurrent Li-GRU model.

It worth mentioning that phone recognition is still a challenging task for spiking neural networks. To the best of our knowledge, only one recent work with recurrent spiking neural networks (Bellec et al., 2019) demonstrates some promising test results on this corpus with a PER of 26.4%. In contrast, our system has achieved significantly lower PER compared to this preliminary study of SNN-based acoustic modeling. However, these results are not directly comparable since the proposed system incorporates both an acoustic and a language model during decoding unlike the system described in Bellec et al. (2019).

The experimental results on the TIMIT phone recognition task can be considered as an initial indicator of the compelling prospects of the SNN-based acoustic modeling. Given that the phone recognition task on TIMIT corpus is simplistic compared to the modern LVCSR tasks, we further compare the

³<https://github.com/kaldi-asr/kaldi/tree/master/egs/timit>;
<https://github.com/kaldi-asr/kaldi/tree/master/egs/librispeech>

TABLE 1 | PER (%) on the TIMIT development and test sets.

Features \ AM	Test	
	Li-GRU (Ravanelli et al., 2018)	RSNN (Bellec et al., 2019)
MFCC (13-dim.)	16.7	26.4
FBANK (13-dim.)	15.8	—
FMLLR	14.9*	—

Features \ AM	Dev	Test	
	ANN	SNN	SNN
MFCC (13-dim.)	17.1	17.8	18.5
FBANK (13-dim.)	16.9	17.3	18.5
MFCC (40-dim.)	17.3	18.2	18.7
FBANK (40-dim.)	16.9	17.8	17.9
FMLLR	15.8	16.5	17.2

The upper panel reports the results of various ANN and SNN architectures from the literatures, and the lower panel presents the results achieved by the ANN and SNN models in this work (AM, acoustic model, the best result to date). The best results given by the speaker-independent features at each column are marked in bold.

ANN and SNN performance on newer corpora designed for LVCSR experiments.

4.2. Low-Resourced ASR on FAME Corpus

In this section, we apply the SNN-based ASR systems to the low-resourced ASR scenario. As summarized in **Table 2**, the word recognition results on the FAME corpus are reported separately for monolingual Frisian (fy), monolingual Dutch (nl) and code-switched (cs) utterances. The overall performance (all) is also included in the rightmost column. Given that 8.5 h Frisian and 3 h of Dutch speech is used during the training phase, we can compare the ASR performance on different subsets, i.e., fy, nl and cs, to identify the variations in the ASR performance for different levels of low-resourcedness. We omit the results on the development set as they follow a similar pattern to the results on the test set.

In this scenario, the SNN acoustic models consistently provide lower WERs than the ANN models for all speech features. Systems with the FBANK features provide lower WERs than those using MFCC features, which is in line with our observations on the TIMIT corpus. The best performance on the test set is obtained using SNN models trained on 40-dimensional FBANK features with an overall WER of 36.9%. In contrast, the ANN model provides a WER of 39.0% for the same setting, which is relatively 5.4% worse than the SNN model. Moreover, the SNN-based acoustic models achieve a relative improvement of 4.7%, 5.2% and 8.2% on the fy, nl and cs subsets of the test set, respectively. These steady improvements in the recognition accuracies highlight the effectiveness of the SNN-based acoustic modeling in scenarios with limited training data compared to the conventional ANN models. The improved ASR performance with SNNs, in the low-resourced setting, may credit to the noisy weight updates derived by the tandem learning framework. It has been recognized that introducing noises into the training

TABLE 2 | WERs (%) achieved on the monolingual and mixed segments of the FAME test set.

		fy	nl	cs	All
# of Frisian words		10,753	0	1,798	12,551
# of Dutch words		0	3,475	306	3,781

Speech features	AM				
FBANK (40-dim.)	Kaldi-ANN (Yilmaz et al., 2016b)	32.4	39.7	49.9	36.2
MFCC (40-dim.)	TDNN-LSTM (Yilmaz et al., 2018)	31.5	39.5	47.9	35.2

MFCC (13-dim.)	ANN	34.6	50.0	49.9	39.9
MFCC (13-dim.)	SNN	33.8	45.3	47.9	38.2
FBANK (13-dim.)	ANN	34.3	47.5	48.1	39.0
FBANK (13-dim.)	SNN	33.1	44.3	46.5	37.3
MFCC (40-dim.)	ANN	35.2	48.4	51.7	40.2
MFCC (40-dim.)	SNN	33.7	44.2	46.9	37.7
FBANK (40-dim.)	ANN	34.4	46.3	49.8	39.0
FBANK (40-dim.)	SNN	32.8	43.9	45.7	36.9
FMLLR	ANN	31.2	42.1	47.2	35.7
FMLLR	SNN	31.5	39.5	46.6	35.2

The upper panel summarizes the number of words from each language subset. The middle panel provides the results of state-of-the-art ANN architectures (Yilmaz et al., 2016b, 2018) for reference purposes and the lower panel presents the results achieved by the ANN and SNN models in this work (AM, acoustic model). The best results given by the speaker-independent features at each column are marked in bold.

stage improves the generalization capability of ANN-based ASR systems (Yin et al., 2015). As a result, the noisy training of the tandem learning is expected to improve the recognition performance in low-resourced scenarios. Further investigation on the impact of this noisy training procedure remains as future work.

4.3. LVCSR Experiments on Librispeech Corpus

In the final set of ASR experiments, we train acoustic models using the official 100 and 360-h training subsets of the Librispeech corpus to compare the recognition performance of ANN and SNN models in a standard LVCSR scenario. As the results given in the middle panel of **Table 3**, for 100 h of training data, the ANN systems perform marginally better than the corresponding SNN systems across all different speech features. The absolute WER differences range from 0.1% to 0.6%. These marginal performance degradations of the SNN models is likely due to the reduced representation power of using discrete spike counts. Nevertheless, these results are promising even when comparing to the state-of-the-art ASR systems using more complex ANN architectures as provided in the upper panel of **Table 3**.

It worth noting that both ANN and SNN systems can take benefit of an increased amount of training data. When increasing the training data from 100 to 360 h, the WERs of the best SNN models reduced from 10.0% (10.3%) to 9.2% (9.4%) for the development (test) sets, respectively. To the best of our knowledge, it is the first time that SNN-based acoustic models

TABLE 3 | WER (%) achieved on the Librispeech development and test sets.

		Train - 100 h			
		Dev		Test	
AM					
Kaldi†	p-norm ANN	9.2		9.7	
PyTorch-Kaldi†	Li-GRU	–		8.6	
Features \ AM		ANN	SNN	ANN	SNN
MFCC		10.3	10.5	10.6	10.9
FBANK		9.6	10.0	10.2	10.6
MFCC (40-dim.)		9.5	10.1	10.0	10.6
FBANK (40-dim.)		9.6	10.2	10.1	10.3
FMLLR		9.2	9.3	9.7	9.9
		Train - 360 h			
		Dev		Test	
Features \ AM		ANN	SNN	ANN	SNN
MFCC		9.2	9.9	9.6	10.3
FBANK		8.6	9.7	9.1	10.0
MFCC (40-dim.)		8.6	9.2	8.9	9.4
FBANK (40-dim.)		8.5	9.4	8.9	9.7
FMLLR		8.4	9.2	8.8	9.7

The upper panel gives the results, with 100-h of training data, reported at the Github repo of Kaldi and PyTorch-Kaldi. The middle and lower panel present the results achieved by ANN and SNN models in this work using 100-h and 360-h of training data, respectively. The best results given by the speaker-independent features in the middle and lower panel are marked in bold. (AM, acoustic model, [†]: reported at Github repo).

have achieved comparable results over the ANN models for LVCSR tasks. These results suggest that SNNs are potentially good candidates for acoustic modeling.

4.4. Energy Efficiency of SNN-Based ASR Systems

In addition to the promising modeling capability, the SNN-based ASR systems can achieve unprecedented performance gain when implemented on the low-power neuromorphic chips. In this section, we shed light on this prospect by comparing the energy efficiency of ANN- and SNN-based acoustic models. Given that data movements are the most energy-consuming operations for data-driven AI applications, we calculate the average synaptic operations on 5 randomly chosen utterances from the TIMIT corpus and report the ratio of average synaptic operations required per feature classification [SynOps(SNN)/SynOps(ANN)]. To investigate the effect of different feature representations, we repeat our analysis on the 40-dimensional MFCC, FBANK, and FMLLR features as summarized in **Table 4** and **Figure 5**.

Taking advantage of the short encoding time window ($N_s = 10$), the sparse neuronal activities are observed for all network layers as shown in **Figure 5**. Among the three features explored in this experiment, it is interesting to note the FMLLR feature achieves the lowest average spike rate. It is likely due to the

TABLE 4 | Comparison of the computational costs between SNN and ANN.

Utterance Index	1	2	3	4	5	Avg. SynOps Ratio
Num. of frames	474	287	274	268	223	
MFCC (40-dim.)	1.71	1.73	1.76	1.71	1.68	1.72
FBANK (40-dim.)	1.08	1.08	1.14	1.09	1.10	1.10
FMLLR	0.67	0.68	0.71	0.66	0.67	0.68

The ratio of their required total synaptic operations [SynOps(SNN)/SynOps(ANN)] is reported. It worth mentioning that ANNs use more costly MAC operations than the AC operations used in the SNNs.

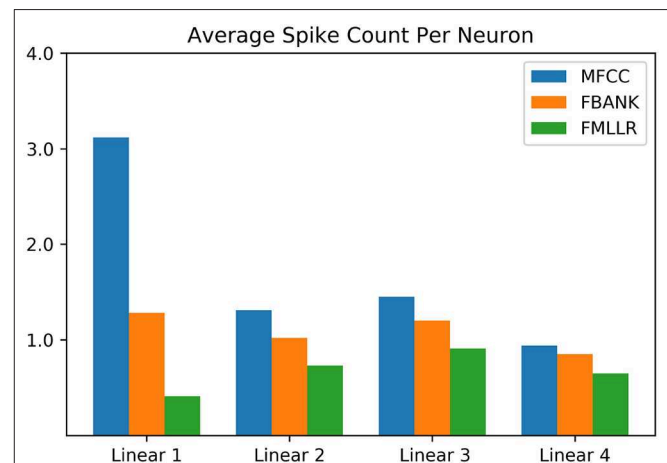


FIGURE 5 | Average spike count per neuron of different SNN layers on the TIMIT corpus. The results of different input features are color-coded. Sparse neuronal activities can be observed in this bar chart.

more discriminative nature of the speaker-dependent feature, while it worth to note that the FMLLR feature is not always available in all ASR scenarios. As provided in **Table 4**, the SNN implementations taking MFCC, FBANK and FMLLR input features require 1.72, 1.10, and 0.68 times synaptic operations to their ANN counterparts, respectively. Although the average number of synaptic operations required for SNNs that using MFCC and FBANK features are slightly higher than the ANNs, the AC operations performed on SNNs are much cheaper than the MAC operations required for ANNs. One recent study on the Global Foundry 28 nm process has revealed that MAC operations are 14 times more costly than AC operations and requires 21 times more chip area (Rueckauer et al., 2017). This study provides some good indicators for the potential energy and chips area savings that can be received from deploying SNNs onto the emerging neuromorphic chips for inference (Merolla et al., 2014; Davies et al., 2018). While the actual energy savings for SNN-based acoustic models are dependent on the chip architectures and materials used, which is beyond the scope of this work.

5. DISCUSSION

The remarkable progress in the automatic speech recognition systems has revolutionized the human-computer interface. The

rapid growing demands of ASR services have raised concerns on computational efficiency, real-time performance, and data security, etc. It, therefore, motivates novel solutions to address all those concerns. As inspired by the event-driven computation that observed in the biological neural systems, we explore using brain-inspired spiking neural networks for large vocabulary ASR tasks. For this purpose, we proposed a novel SNN-based ASR framework, wherein the SNN is used for acoustic modeling and map the frame-level features into a set of acoustic units. These frame-level outputs will further integrate the word-level information from the corresponding language model to find the most likely word sequence corresponding to the input speech signal.

5.1. Superior Speech Recognition Performance With SNNs

The phone and word recognition experiments on the well-known TIMIT and Librispeech benchmarks have demonstrated the promising modeling capacity of SNN acoustic models and their applicability to different input features. These preliminary results have shown that the recognition performance of SNNs is either comparable or slightly worse than the ANNs with the same network architecture on the TIMIT and Librispeech benchmarks. A possible reason for this performance degradation is the reduced representation power of the discrete neural representation (i.e., spike counts) as compared to the continuous floating-point representation of the ANNs (Wu et al., 2019c). This performance gap could potentially be closed by extending the encoding window N_s of SNNs. Moreover, the recognition performance of ANN and SNN models in a low-resourced scenario is also investigated. In this scenario, the SNN acoustic models outperform the conventional ANNs that could be attributed to the noisy training of the tandem learning framework, wherein error gradients of the SNN layers are approximated from the coupled ANN layers.

The neural encoding scheme adopted in this work allows input features to be encoded inside a short encoding time window for rapid processing by SNNs. It is attractive for the time-synchronous ASR tasks that require real-time performance. The preliminary study of the energy efficiency on the TIMIT corpus reveals attractive energy and chip area savings, as compared to the equivalent ANNs, can be achieved when deploying the offline trained SNNs onto neuromorphic chips. The recent study of a keyword spotting task on the Loihi neuromorphic research chip (Blouw et al., 2019) has also demonstrated the compelling energy savings, real-time performance and good scalability of emerging NC architectures over conventional low-power AI chips designed for ANNs.

5.2. Development of SNN-Based ASR Systems

The active development of open-source software toolkits plays a significant role in the rapid progress of ASR research, instances include the Kaldi (Povey et al., 2011) and ESPnet (Watanabe et al., 2018). In this work, we demonstrate that state-of-the-art SNN acoustic models can be easily developed

in PyTorch and integrated into the PyTorch-Kaldi Speech Recognition Toolkit (Ravanelli et al., 2019). This software toolkit integrates the efficiency of Kaldi and the flexibility of PyTorch, therefore, it can support the rapid development of SNN-based ASR systems.

5.3. Future Directions

The recurrent neural networks have shown great modeling capability for temporal signals by exploring long temporal context information in the input signals (Graves and Jaitly, 2014). As future work, we will explore the recurrent networks of spiking neurons for large-vocabulary ASR tasks to further improve the recognition performance.

The substantial research efforts are devoted to reducing the computational cost and memory footprint of ANNs during inference, instances include network compression (Han et al., 2015), network quantization (Courbariaux et al., 2016; Zhou et al., 2016) and knowledge distillation (Hinton et al., 2015). While the computational paradigm underlying the efficient biological neural networks is fundamentally different from ANNs and hence fosters enormous potentials for neuromorphic computing architectures. Furthermore, grounded on the same connectionism principle, the information of both ANN and SNN are encoded in the network connectivity and connection strength. Therefore, SNN can also take benefits from these early research works on the network compression and quantization of ANNs to further reduce its memory footprint and computation cost (Deng et al., 2019).

The event-driven silicon cochlea audio sensors (Liu et al., 2014) are designed to mimic the functional mechanism of human cochlea and transform input audio signals into spiking events. Given temporally sparse information is transmitted in the surrounding environment, these sensors have shown greater coding efficiency than conventional microphone sensors (Liu et al., 2019). There are some interesting preliminary ASR studies explore the input spiking events captured by these silicon cochlea sensors (Acharya et al., 2018; Anumula et al., 2018). Additionally, Dominguez-Morales et al. (2018) have proposed a fully SNN-based framework for voice commands recognition, wherein the event-driven silicon cochlea audio sensor is directly interfaced with the SpiNNaker neuromorphic processor through the Address-Event Representation protocol (AER). Notably, a buffering layer is introduced to ensure real-time performance. However, the scale of the ASR tasks explored in these studies is relatively small comparing to modern ASR benchmarks due to the limited availability of event-based ASR corpora. Pan et al. (2020) recently proposed an efficient and perceptually motivated auditory neural encoding scheme to encode the large-scale ASR corpora collected by microphone sensors into spiking events. With this encoding scheme, approximately 50% spiking events can be reduced with negligible interference to the perceptual quality of inputs audio signals. Taking benefits from these earlier research on the neuromorphic auditory front-end, we are expecting to further improve the energy efficiency of SNN-based ASR systems.

The promising initial results demonstrated by the SNN-based large vocabulary ASR systems in this work is the

first step toward a myriad opportunities for the integration of state-of-the-art ASR engines into mobile and embedded devices with power restrictions. In the long run, the SNN-based ASR systems are expected to take benefits from ever-growing research on novel neuromorphic auditory front-end, SNN architectures, neuromorphic computing architectures and ultra-low-power non-volatile memory devices to further improve the computing performance.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: <https://catalog.ldc.upenn.edu/LDC93S1>; <https://www.openslr.org/12>; <https://repository.ubn.ru.nl/handle/2066/162244>.

REFERENCES

- Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* 22, 1533–1545. doi: 10.1109/TASLP.2014.2339736
- Acharya, J., Patil, A., Li, X., Chen, Y., Liu, S.-C., and Basu, A. (2018). A comparison of low-complexity real-time feature extraction for neuromorphic speech recognition. *Front. Neurosci.* 12:160. doi: 10.3389/fnins.2018.00160
- Anumula, J., Neil, D., Delbruck, T., and Liu, S. C. (2018). Feature representations for neuromorphic audio spike streams. *Front. Neurosci.* 12:23. doi: 10.3389/fnins.2018.00023
- 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), 787–797.
- Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2019). A solution to the learning dilemma for recurrent networks of spiking neurons. *bioRxiv [Preprint]*. doi: 10.1101/738385. Available online at: <https://graz.pure.elsevier.com/en/publications/a-solution-to-the-learning-dilemma-for-recurrent-networks-of-spik>
- Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2019). “Benchmarking keyword spotting efficiency on neuromorphic hardware,” in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop* (Albany, NY: ACM), 1.
- 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
- Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2016). “Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,” in *Proceedings of the ICASSP* (Shanghai: IEEE), 4960–4964. doi: 10.1109/ICASSP.2016.7472621
- Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. *arXiv:1602.02830*.
- Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans. Audio Speech Lang. Process.* 20, 30–42. doi: 10.1109/TASL.2011.2134090
- 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
- Davis, S., and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. Acoust.* 28, 357–366. doi: 10.1109/TASSP.1980.1163420
- 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:1911.00822*.

AUTHOR CONTRIBUTIONS

JW and EY designed and conducted all the experiments. All authors contributed to the results interpretation and writing.

FUNDING

This research was supported by Programmatic Grant No. A1687b0033 from the Singapore Government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain). JW was also partially supported by the Zhejiang Lab's International Talent Fund for Young Professionals and the Zhejiang Lab (No.2019KC0AB02). HL was also partially supported by U Bremen Excellence Chairs program (2019–2022), Germany.

- 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), 1–8.
- Dominguez-Morales, J. P., Liu, Q., James, R., Gutierrez-Galan, D., Jimenez-Fernandez, A., Davidson, S., et al. (2018). “Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach,” in *2018 International Joint Conference on Neural Networks (IJCNN)* (Rio de Janeiro), 1–8.
- Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2012). Overview of the SpiNNaker system architecture. *IEEE Trans. Comput.* 62, 2454–2467. doi: 10.1109/TC.2012.142
- Gales, M. J. F. (1998). Maximum likelihood linear transformations for hmm-based speech recognition. *Comput. Speech Lang.* 12, 75–98. doi: 10.1006/csla.1998.0043
- Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., Dahlgren, N. L., et al. (1993). *TIMIT Acoustic-Phonetic Continuous Speech Corpus (LDC93S1)*. Philadelphia, PA: Linguistic Data Consortium.
- Gerstner, W., and Kistler, W. M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge: Cambridge University Press.
- Graves, A., and Jaitly, N. (2014). “Towards end-to-end speech recognition with recurrent neural networks,” in *Proceedings of the 31st International Conference on Machine Learning (ICML)* (Beijing: ACM), 1764–1772.
- Graves, A., Mohamed, A., and Hinton, G. (2013). “Speech recognition with deep recurrent neural networks,” in *Proceedings of the ICASSP* (Vancouver, BC), 6645–6649.
- Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2016). Lstm: a search space odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 2222–2232. doi: 10.1109/TNNLS.2016.2582924
- Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv:1510.00149*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). “Deep residual learning for image recognition,” in *Proceedings of the IEEE CVPR* (Las Vegas, NV), 770–778. doi: 10.1109/CVPR.2016.90
- He, T., Fan, Y., Qian, Y., Tan, T., and Yu, K. (2014). “Reshaping deep neural network for fast decoding by node-pruning,” in *Proceedings of the ICASSP* (Vancouver, BC), 245–249. doi: 10.1109/ICASSP.2014.6853595
- Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-R., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process. Mag.* 29, 82–97. doi: 10.1109/MSP.2012.2205597
- Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv:1503.02531*.
- Holmberg, M., Gelbart, D., Ramacher, U., and Hemmert, W. (2005). “Automatic speech recognition with neural spike trains,” in *INTERSPEECH* (Lisbon), 1253–1256.

- Hu, Y., Tang, H., Wang, Y., and Pan, G. (2018). Spiking deep residual network. *arXiv:1805.01352*.
- Hwang, M.-Y., and Huang, X. (1993). Shared-distribution hidden markov models for speech recognition. *IEEE Trans. Speech Audio Process.* 1, 414–420. doi: 10.1109/89.242487
- Kröger, B. J., Kannampuzha, J., and Neuschaefer-Rube, C. (2009). Towards a neurocomputational model of speech production and perception. *Speech Commun.* 51, 793–809. doi: 10.1016/j.specom.2008.08.002
- Lang, K. J., Waibel, A. H., and Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Netw.* 3, 23–43. doi: 10.1016/0893-6080(90)90044-L
- Laughlin, S. B., and Sejnowski, T. J. (2003). Communication in neuronal networks. *Science* 301, 1870–1874. doi: 10.1126/science.1089662
- Lee, K. (1990). Context-independent phonetic hidden markov models for speaker-independent continuous speech recognition. *IEEE Trans. Acoust.* 38, 599–609. doi: 10.1109/29.52701
- Lei, X., Senior, A. W., Gruenstein, A., and Sorensen, J. S. (2013). “Accurate and compact large vocabulary speech recognition on mobile devices,” in *Proceedings of the INTERSPEECH* (Vancouver, BC), 662–665.
- Liaw, J. S., and Berger, T. W. (1998). “Robust speech recognition with dynamic synapses,” in *IEEE International Joint Conference on Neural Networks Proceedings (IJCNN)*, Vol. 3 (Anchorage, AK), 2175–2179.
- Lippmann, R. P. (1989). Review of neural networks for speech recognition. *Neural Comput.* 1, 1–38. doi: 10.1162/neco.1989.1.1.1
- Liu, S.-C., Rueckauer, B., Ceolini, E., Huber, A., and Delbruck, T. (2019). Event-driven sensing for efficient perception: vision and audition algorithms. *IEEE Signal Process. Mag.* 36, 29–37. doi: 10.1109/MSP.2019.2928127
- Liu, S. C., van Schaik, A., Minch, B. A., and Delbruck, T. (2014). Asynchronous binaural spatial audition sensor with 2x64x4 channel output. *IEEE Trans. Biomed. Circuits Syst.* 8, 453–464. doi: 10.1109/TBCAS.2013.2281834
- Loiselle, S., Rouat, J., Pressnitzer, D., and Thorpe, S. (2005). “Exploration of rank order coding with spiking neural networks for speech recognition,” in *IEEE International Joint Conference on Neural Networks (IJCNN)*, Vol. 4 (Montréal, QC), 2076–2080.
- McGraw, I., Prabhavalkar, R., Alvarez, R., Arenas, M. G., Rao, K., Rybach, D., et al. (2016). “Personalized speech recognition on mobile devices,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Shanghai), 5955–5959.
- 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
- Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Comput. Speech Lang.* 16, 69–88. doi: 10.1006/csla.2001.0184
- Monroe, D. (2014). Neuromorphic computing gets ready for the (really) big time. *Commun. ACM* 57, 13–15. doi: 10.1145/2601069
- Myers-Scotton, C. (1989). Codeswitching with English: types of switching, types of communities. *World Englishes* 8, 333–346. doi: 10.1111/j.1467-971X.1989.tb00673.x
- Näger, C., Storck, J., and Deco, G. (2002). Speech recognition with spiking neurons and dynamic synapses: a model motivated by the human auditory pathway. *Neurocomputing* 44–46, 937–942. doi: 10.1016/S0925-2312(02)00494-0
- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *arXiv: 1901.09948*.
- Pan, Z., Chua, Y., Wu, J., Zhang, M., Li, H., and Ambikairajah, E. (2020). An efficient and perceptually motivated auditory neural encoding and decoding algorithm for spiking neural networks. *Front. Neurosci.* 13:1420. doi: 10.3389/fnins.2019.01420
- 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: IEEE), 1–8.
- Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). “Librispeech: an ASR corpus based on public domain audio books,” in *Proceedings of the ICASSP* (South Brisbane, QLD), 5206–5210.
- 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* 32, eds H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett (Vancouver, BC: Curran Associates, Inc.), 8026–8037.
- Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities & challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774
- Povey, D., Cheng, G., Wang, Y., Li, K., Xu, H., Yarmohammadi, M., et al. (2018). “Semi-orthogonal low-rank matrix factorization for deep neural networks,” in *Proceedings of the Interspeech* (Hyderabad), 3743–3747. doi: 10.21437/Interspeech.2018-1417
- Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., et al. (2011). “The kaldi speech recognition toolkit,” in *IEEE ASRU* (Hawaii).
- Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., et al. (2012). “Generating exact lattices in the WFST framework,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Kyoto), 4213–4216.
- Ravanelli, M., Brakel, P., Omologo, M., and Bengio, Y. (2018). Light gated recurrent units for speech recognition. *IEEE Trans. Emerg. Top. Comput. Intell.* 2, 92–102. doi: 10.1109/TETCI.2017.2762739
- Ravanelli, M., Parcollet, T., and Bengio, Y. (2019). “The pytorch-kaldi speech recognition toolkit,” in *Proceedings of the ICASSP* (Brighton, UK), 6465–6469.
- 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
- Sainath, T. N., Kingsbury, B., Sindhwan, V., Arisoy, E., and Ramabhadran, B. (2013). “Low-rank matrix factorization for deep neural network training with high-dimensional output targets,” in *Proceedings of the ICASSP* (Vancouver, BC), 6655–6659.
- Sainath, T. N., and Parada, C. (2015). “Convolutional neural networks for small-footprint keyword spotting,” in *Proceedings of the INTERSPEECH* (Dresden), 1478–1482.
- 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., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature* 550:354. doi: 10.1038/nature24270
- Tang, J., Yuan, F., Shen, X., Wang, Z., Rao, M., He, Y., et al. (2019). Bridging biological and artificial neural networks with emerging neuromorphic devices: fundamentals, progress, and challenges. *Adv. Mater.* 31:1902761. doi: 10.1002/adma.201902761
- Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002
- Tavanaei, A., and Maida, A. (2017a). “Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals,” in *International Conference on Neural Information Processing* (Guangzhou: Springer), 899–908.
- Tavanaei, A., and Maida, A. (2017b). A spiking network that learns to extract spike signatures from speech signals. *Neurocomputing* 240, 191–199. doi: 10.1016/j.neucom.2017.01.088
- Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., et al. (2018). Espnet: End-to-end speech processing toolkit. *arXiv: 1804.00015*. doi: 10.21437/Interspeech.2018-1456
- Watanabe, S., Hori, T., Kim, S., Hershey, J. R., and Hayashi, T. (2017). Hybrid CTC/attention architecture for end-to-end speech recognition. *IEEE J. Sel. Top. Signal Process.* 11, 1240–1253. doi: 10.1109/JSTSP.2017.2763455
- 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), 1–8.
- Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2019c). A tandem learning rule for efficient and rapid inference on deep spiking neural networks. *arXiv:1907.01167*.
- 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. (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, J., Pan, Z., Zhang, M., Das, R. K., Chua, Y., and Li, H. (2019b). "Robust sound recognition: a neuromorphic approach," in *Proceedings of the Interspeech 2019* (Graz), 3667–3668.
- Wu, M., Panchapagesan, S., Sun, M., Gu, J., Thomas, R., Prasad Vitaladevuni, S. N., et al. (2018). "Monophone-based background modeling for two-stage on-device wake word detection," in *Proceedings of the ICASSP* (Calgary, AB), 5494–5498.
- Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. (2018). Direct training for spiking neural networks: faster, larger, better. *arXiv: 1809.05793*.
- Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M. L., Stolcke, A., et al. (2017). Toward human parity in conversational speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* 25, 2410–2423. doi: 10.1109/TASLP.2017.2756440
- Xue, J., Li, J., and Gong, Y. (2013). "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proceedings of the Interspeech* (Lyon), 2365–2369.
- Yilmaz, E., Andringa, M., Kingma, S., Van der Kuip, F., Van de Velde, H., Kampstra, F., et al. (2016a). "A longitudinal bilingual Frisian-Dutch radio broadcast database designed for code-switching research," in *Proceedings of the LREC* (Portorož), 4666–4669.
- Yilmaz, E., van den Heuvel, H., and van Leeuwen, D. (2016b). "Code-switching detection using multilingual DNNs," in *2016 IEEE Spoken Language Technology Workshop (SLT)* (San Diego, CA), 610–616.
- Yilmaz, E., Van den Heuvel, H., and Van Leeuwen, D. A. (2018). "Acoustic and textual data augmentation for improved ASR of code-switching speech," in *Proceedings of the INTERSPEECH* (Hyderabad), 1933–1937.
- Yin, S., Liu, C., Zhang, Z., Lin, Y., Wang, D., Tejedor, J., et al. (2015). Noisy training for deep neural networks in speech recognition. *EURASIP J. Audio Speech Music Process.* 2015:2. doi: 10.1186/s13636-014-0047-0
- Yu, D., and Deng, L. (2015). *Automatic Speech Recognition: A Deep Learning Approach. Signals and Communication Technology*. Lisbon: Springer.
- Zehetner, A., Hagmuller, M., and Pernkopf, F. (2014). "Wake-up-word spotting for mobile systems," in *Proceedings of the EUSIPCO*, 1472–1476.
- Zhang, M., Wu, J., Chua, Y., Luo, X., Pan, Z., Liu, D., et al. (2019). "Mpd-al: an efficient membrane potential driven aggregate-label learning algorithm for spiking neurons," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33 (Hawaii), 1327–1334. doi: 10.1609/aaai.v33i01.33011327
- 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
- Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. (2016). Dorefanet: training low bitwidth convolutional neural networks with low bitwidth gradients. *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.

Copyright © 2020 Wu, Yilmaz, Zhang, Li and Tan. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Noise Helps Optimization Escape From Saddle Points in the Synaptic Plasticity

Ying Fang^{1,2,3}, Zhao Fei Yu⁴ and Feng Chen^{1,2,3*}

¹ Department of Automation, Center for Brain-Inspired Computing Research, Tsinghua University, Beijing, China, ² Beijing Innovation Center for Future Chip, Beijing, China, ³ Beijing Key Laboratory of Security in Big Data Processing and Application, Beijing, China, ⁴ National Engineering Laboratory for Video Technology, School of Electronics Engineering and Computer Science, Peking University, Beijing, China

OPEN ACCESS

Edited by:

Huajin Tang,
Zhejiang University, China

Reviewed by:

Xiurui Xie,
Agency for Science, Technology and
Research (A*STAR), Singapore
David Kappel,
Dresden University of Technology,
Germany

*Correspondence:

Feng Chen
chenfeng@mail.tsinghua.edu.cn

Specialty section:

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

Received: 29 November 2019

Accepted: 23 March 2020

Published: 29 April 2020

Citation:

Fang Y, Yu Z and Chen F (2020) Noise
Helps Optimization Escape From
Saddle Points in the Synaptic
Plasticity. *Front. Neurosci.* 14:343.
doi: 10.3389/fnins.2020.00343

Numerous experimental studies suggest that noise is inherent in the human brain. However, the functional importance of noise remains unknown. In particular, from a computational perspective, such stochasticity is potentially harmful to brain function. In machine learning, a large number of saddle points are surrounded by high error plateaus and give the illusion of the existence of local minimum. As a result, being trapped in the saddle points can dramatically impair learning and adding noise will attack such saddle point problems in high-dimensional optimization, especially under the strict saddle condition. Motivated by these arguments, we propose one biologically plausible noise structure and demonstrate that noise can efficiently improve the optimization performance of spiking neural networks based on stochastic gradient descent. The strict saddle condition for synaptic plasticity is deduced, and under such conditions, noise can help optimization escape from saddle points on high dimensional domains. The theoretical results explain the stochasticity of synapses and guide us on how to make use of noise. In addition, we provide biological interpretations of proposed noise structures from two points: one based on the free energy principle in neuroscience and another based on observations of *in vivo* experiments. Our simulation results manifest that in the learning and test phase, the accuracy of synaptic sampling with noise is almost 20% higher than that without noise for synthesis dataset, and the gain in accuracy with/without noise is at least 10% for the MNIST and CIFAR-10 dataset. Our study provides a new learning framework for the brain and sheds new light on deep noisy spiking neural networks.

Keywords: noise, strict saddle, synaptic sampling, synaptic plasticity, free energy

1. INTRODUCTION

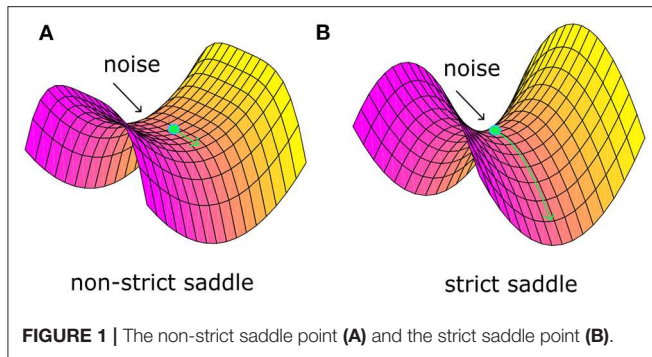
It has been observed that noise permeates everywhere in the nervous system and affects all aspects of brain function (Mori and Kai, 2002; Fellous et al., 2004; Faisal et al., 2008). On the other hand, it has been proposed that action, perception, and learning in the brain such as attention, memory, neural coding, and evolution, can be understood as an optimization process (Friston, 2010). Both noise and optimization are prevalent in the nervous system. So, is there a close relationship between the two? What precisely is the nature of noise that helps the brain compute optimally? In this paper, we argue that typically the answer is YES.

In recent years, many studies have provided insight into which noise structures are present, and how noise affects the structure and function of the nervous system. As far as we know, there are mainly three noise models in stochastic neural circuit computations. The first model is based on the leaky integrate-and-fire (LIF) model. Although LIF models have been extensively applied in biological spiking neural networks, they are still deterministic. Some researchers add a Brownian noise on the potential of IF neurons for better agreement with experimental observations (Soula et al., 2004; Burkitt, 2006; Cessac, 2010, 2011). Brownian noise helps them characterize generic behaviors by exploring a large number of parameters. However, these researchers did not further study other benefits of noise. The second model is based on the mean-field theory. The mean field of one neuron represents its effect on the whole neural network. They add noise in the neuron's behavior by assuming a neuron has an instantaneous firing probability in any time step (Galves and Löcherbach, 2013, 2016; Larremore et al., 2014; Duarte et al., 2015). This firing probability $P_F(V)$ is a function of membrane potential V . The mean field depends on the synaptic weights and firing probabilities $P_F(V)$ from interconnected neurons. Therefore, they simplify the analysis and simulation of noisy spiking neural networks in the mean-field calculation. However, this model groups all sources of noise into a single firing function and is therefore agnostic about the origin of noise. As a result, it is difficult to decompose explicit noise terms from the model, which is a bad thing for the mathematical analysis of noise. The third model is based on sampling. General results from statistical learning theory suggest that both brain computations and brain plasticity should be understood as probabilistic inference (Knill and Pouget, 2004; Pouget et al., 2013). These results have provided insight into how noise plays an essential role in the networks of spiking neurons. Based on Boltzmann machines, Maass (2014) propose that knowledge can be stored in probabilistic distributions of network states, and noise enables networks of spiking neurons to carry out probabilistic inference through MCMC sampling. The sample of this model is the state of neurons and the noise results from the ion channels of excitable membranes. Based on Langevin sampling, Kappel et al. (2015, 2018) analyzed continuously ongoing synapse dynamics and noise endows networks to compensate for internal and external changes automatically in the local plasticity mechanisms. The sample of this model is the state of synapses, and the noise results from the synaptic transmission. In the work by Kappel et al. (2018) in particular, they discuss the impact of different temperatures on learning performance, where the strength of stochasticity can be scaled by the temperature. Results show that good performance was achieved for a range of temperature values and temperatures that were too low (such as without noise) or too high impaired learning. They provide a short explanation through the perspective of an analogy of simulated annealing. However, they did not provide a rigorous theoretical analysis for the noise mechanism. In conclusion, although researchers using the sampling model have claimed that the benefit of noise is a functional part of sampling, to perform probabilistic inference, they do not provide a detailed mathematical analysis of noise and do not study which noise structure is involved,

or how it enhances the computation power of spiking neural networks.

In summary, in theoretical neuroscience research, the extent to which noise is biologically present and how noise improves computation performance in the brain has rarely been addressed. Based on the synaptic sampling model (Kappel et al., 2015), we give a detailed mathematical analysis of noise in spiking neural networks and try to explain why our brain benefits from noise. Here, we can generally assume that noise type is fluctuations in synaptic transmission because the proposed noise has an important role in synaptic plasticity. There are many sources of noise in synaptic transmission, such as stochastic molecular diffusion (Holcman et al., 2005), short-term plasticity (Abbott and Regehr, 2004), and synaptic neurotransmitter release (Branco and Staras, 2009). Therefore, we make no assumptions about the concrete sources of noise. Next, we will sketch the noise mechanism and try to bridge the gap between neuromorphic computing and machine learning.

According to the free-energy principle in neuroscience, we propose a biologically plausible noise structure and prove that such noise helps optimization escape from bad saddle points in the brain computation and brain plasticity. First, we propose that one of the essential roles of noise is to improve optimization and prove that the noise mechanisms of improving optimization satisfy the strict-saddle condition of spiking neural networks. The main bottleneck in optimization is that gradient updates are trapped in exponentially more saddle points instead of local minima (Fedorov and Williams, 2007; Dauphin et al., 2014). Under the so-called strict saddle property, gradient descent with noise will escape from bad saddle points and lead to efficient optimization (Ge et al., 2015). The importance of adding perturbations for efficient non-convex optimization has been justified in many machine applications, including deep learning (Du et al., 2017; Jin et al., 2017). We prove that such noises make spiking neural networks satisfy strict saddle properties by changing the curvature of the landscape in the network parameter space, especially in the area near the saddle points. In other words, noise helps spiking neural networks build appropriate Hessian constructions, and optimization can utilize enriched curvature information of the node in the direction without ever calculating or storing the Hessian itself. Second, the proposed noise in the brain theoretically minimizes the free energy of noise signals. In neuroscience, any self-organizing system at equilibrium must minimize its free-energy to resist disorder (Friston, 2010; Joffily and Coricelli, 2013; Apps and Tsakiris, 2014; Colombo and Wright, 2018). Since free energy can be expressed by long-term average self-information of sensory signals, such as mean square error, we prove that such a particular form of noise comes from minimum mean square error estimation. Third, such noise satisfies the fundamental biological characteristic. It is popular to use Brownian motion to describe continuous random fluctuations in spiking activities (Tuckwell, 1988; Cateau and Fukai, 2003; Cateau and Reyes, 2006; Nobuaki et al., 2009). Compared with traditional Brownian motion, the difference is that the standard deviation of our noise has a positive correlation with dendritic spine size. Moreover, it has been observed that larger spines show the most diverse



changes in CA1 pyramidal neurons (Nobuaki et al., 2009). Our noise has a greater standard deviation for larger spines and is hence biologically plausible. Finally, our noisy spiking neural network can also be extended to multi-layer neural networks and obtains better performance. It is generally believed that deep neural networks can learn more complex representations and has shown remarkable success in diverse fields (LeCun et al., 2015). As an example, we realize three-layer noisy spiking neural networks based on gradient back-propagation, which provides a possibility for the realization of large-scale deep networks.

This paper is organized as follows. In section 2, we introduce some complex concepts, such as synaptic sampling and the saddle point problem in non-convex optimization. In section 3, we demonstrate how noise helps optimization escape from saddle points in the neural dynamics and give a proof of sketch on the “strict saddle” condition for synaptic sampling. In section 4, we will explain why the proposed noise structure is biologically plausible from two points: origin from the free-energy principle in neuroscience and consistency with biological observation. In section 5, we derive the learning rule for multi-layer spiking neural networks based on gradient back-propagation for better learning abilities. In section 6, numerical simulations are presented and analyzed. In section 7, we highlight the main contributions of this work and discuss some related open problems. Detailed theorem derivations are deferred to in **Appendices 1, 2, 3 (Supplementary Presentation 1)**.

2. PRELIMINARIES

2.1. Spiking Neural Networks and Hebb Rule

Spiking neural networks (SNNs) is one of the brain-inspired computing models. Its spike-based coding tends to represent more complex information due to spatio-temporal dynamics. In addition, its computation occurs only when the unit in networks receives a spike signal. Such event-driven property is consistent with emerging neuromorphic hardware. Therefore, SNNs have great potential for energy-efficient processing on neuromorphic hardware (Deneve, 2008; Merolla et al., 2014).

In experimental neuroscience, changes of synaptic strength are called synaptic plasticity. The Hebb rule describes how the strength between pre- and postsynaptic neurons should be

modified in synaptic plasticity. It is informally summarized as “Cells that fire together, wire together.” Spike-timing-dependent plasticity (STDP) is one of the Hebbian learning methods. The strength and direction of learning depends on the timing difference between pre- and postsynaptic spikes (Bi and Poo, 1998; Gerstner and Kistler, 2002; Sjöström et al., 2002).

2.2. Synaptic Sampling

Network plasticity by maximum likelihood has been studied in many ways. The inputs \mathbf{x} impinge on the network from its environment. By maximizing the likelihood of the inputs, the network parameters θ are adjusted to encode the input information. That is to say, maximizing the likelihood, is to fit the resulting internal model to the inputs as best as possible. However, the model tends to produce overfitting, thereby reducing generalization capabilities. Furthermore, without any prior distribution, it responds slowly to perturbations. The solution to such a challenge is how the posterior distribution of weights can be represented and learned in neural dynamics. Based on stochastic differential equations, Kappel et al. (2015) solve this challenge by sampling from posterior distribution $p_N(\theta|\mathbf{x})$. This model defined by Equation (1) is referred to as synaptic sampling. Furthermore, they only understand noise as a functional aspect of learning because it helps the network sample from posterior distributions. However, when this model is used for classification with a standard Gaussian noise, it is difficult to find a reasonable minimum due to the saddle point problem, which will be introduced next.

$$d\theta_{ki} = b \left(\frac{\partial \log p_S(\theta)}{\partial \theta_{ki}} + \frac{\partial \log p_N(\mathbf{x}|\theta)}{\partial \theta_{ki}} \right) dt + b dW_{ki} \quad (1)$$

2.3. Saddle Point Problem

Critical points (i.e., minima, maxima, saddle points) are often surrounded by error plateaus of small curvatures, and hence are attractive for the gradient-based learning process. However, as gradient-based algorithms only depend on gradient information, they often mistake saddle points for local minima or maxima. Moreover, it is generally believed that a high-dimensional error functions are likely to have saddle points rather than local minima because the number of saddle points dominate over local minimum exponentially with increasing dimensions (Fyodorov and Williams, 2007; Dauphin et al., 2014). Therefore, gradient-based algorithms are particularly sensitive to saddle point problems.

Recently, Ge et al. (2015) identified a “strict saddle” condition, which guarantees that stochastic gradient descent can escape from the saddle points quickly (see Theorem 6 in work Ge et al., 2015). Note that a twice differentiable function $f(\theta)$ is a strict saddle, if all its local maxima have $\nabla^2 f(\theta) < 0$ and all its other stationary points satisfy $\lambda_{\max}(\nabla^2 f(\theta)) > 0$. Note that λ_{\max} defines the maximum eigenvalue. In fact, the “strict saddle” condition guarantees that there will be at least one descent direction in the small neighborhood of saddle points, not a plain area. For example, **Figure 1A** shows one non-strict saddle point. The area around it is plain and it would be very tough for optimization to escape from such a bottleneck even with noise.

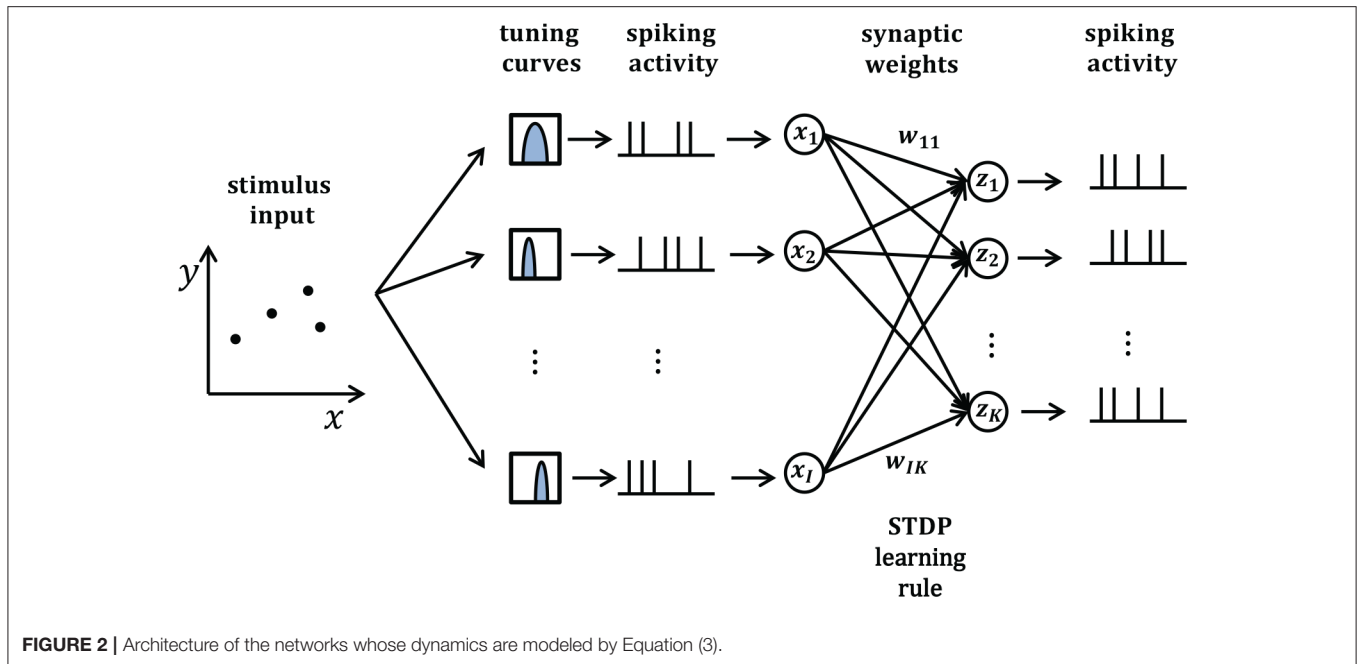


TABLE 1 | Definitions of the main mathematical symbols used in this paper.

\mathbf{x}^n	Vector of the n_{th} input variables $\{x_1^n, \dots, x_I^n\}$
\mathbf{y}^n	Vector of the n_{th} hidden variables $\{y_1^n, \dots, y_J^n\}$
\mathbf{z}^n	Vector of the n_{th} output variables $\{z_1^n, \dots, z_K^n\}$
\mathbf{h}^n	Vector of the label $\{h_1^n, \dots, h_K^n\}$
\mathbf{w}	Vector of all synaptic weights $w_{ki} = e^{(\theta_{ki} - \theta_0)}$ from input neuron i to network neuron k
θ	Vector of all synaptic parameters $\{\theta_{ki}, k \leq K, i \leq I\}$
$p_S(\theta)$	Structural constraints following $\mathcal{N}(\mu, \sigma^2)$
$p_N(J \theta)$	Likelihood function with the form of cross-entropy $\log p_N(J \theta) = \sum_{n=1}^N \sum_{k=1}^K \Theta(h_k^n) \log p(z_k^n \mathbf{x}^n, \theta)$
$p_N(\mathbf{x}^n \theta)$	Poissonian distributions of spikes parameterized by $\alpha e^{w_{ki}}$
dW_{ki}	Stochastic time course of the parameter θ_{ki}
$\Theta(h_k^n)$	Heaviside step function
$S_k(t)$	The spike train of the neuron z_k

Figure 1B shows one strict saddle point. There are at least one descent direction and it will take little time to escape with noise. Based on the above theory, we propose a sufficient condition that noise should satisfy and argue that noise plays a critical role in the brain optimization process.

3. “STRICT SADDLE” CONDITION FOR SYNAPTIC SAMPLING

In this section, we will take synaptic sampling neural networks as an example and demonstrate how noise improves optimization in neural dynamics. We study the effect of noise on the synaptic sampling defined in Equation (1) for classification. As **Figure 2** shows, in the spike-based Winner-Take-All (WTA) circuit, input

neurons tune n_{th} stimulus to 200-ms long spiking activities \mathbf{x}^n according to tuning curves. Given the n_{th} stimulus, the input $x_i(t)$ is expressed by the summation of excitatory postsynaptic potentials (EPSPs) on neurons i in Equation (2).

$$x_i(t) = \sum_f \epsilon(t - t_i^{(f)}) \quad (2)$$

where $t_i^{(f)}$ denotes the spike times of input neuron i and ϵ is the response kernel for spike input, i.e., the shape of the EPSP (Kappel et al., 2015). The corresponding instantaneous firing rate $\rho_k(t)$ of neurons k depend exponentially on the membrane potential $u_k(t)$. In this case, neural networks output a 200-ms spiking pattern \mathbf{z}^n and the neuron which spikes most indicates the possible label. Synaptic sampling is then applied to $K \times I$ synapses. The learning goal in Equation (1) becomes the posterior distribution $p^*(\theta|J)$ defined by $p_S(\theta) * p_N(J|\theta)$. $p_N(J|\theta)$ measures the degree of network fitting to the classification. The detailed definition is shown in **Table 1**. The synaptic sampling rule (Equation 1) yields for this model.

$$d\theta_{ki} = b \left(\frac{1}{\sigma^2} (\mu - \theta_{ki}) + \sum_{n=1}^N w_{ki} (x_i^n - \alpha e^{w_{ki}}) \left(\Theta\{h_k^n\} - S_k(t) \right) \right) dt + b dW_{ki} \quad (3)$$

In Equation (3), the component $(x_i^n - \alpha e^{w_{ki}}) (\Theta\{h_k^n\} - S_k(t))$ of likelihood differential term is a simplified version of STDP (spike timing-dependent plasticity) (Habenschuss et al., 2013; Nessler et al., 2013). Biological studies on STDP show that the timing difference between pre- and post-synaptic spikes decide the strength and direction of learning (Bi and Poo, 1998;

Sjöström et al., 2002). When a presynaptic spike comes before a postsynaptic spike, x_i is large and the term $(x_i^n - \alpha e^{w_{ki}})$ is positive at the time of the postsynaptic spike. Therefore, the term $(x_i^n - \alpha e^{w_{ki}})$ leads to potentiation. When a presynaptic spike comes after a postsynaptic spike, w_{ki} is large and the term $(x_i^n - \alpha e^{w_{ki}})$ is negative at the time of the postsynaptic spike. Therefore, the term $(x_i^n - \alpha e^{w_{ki}})$ leads to depression. In addition, the intensity of potentiation is inversely correlated with synaptic weights. It is consistent with experimental STDP studies (Habenschuss et al., 2013; Nessler et al., 2013).

We show that when the noise takes a certain form, synaptic sampling networks for classification satisfy the “strict saddle” condition and leads to efficient optimization. Note that if the noise is just standard normal distribution, which is a popular choice for stochastic differential equations, networks will not satisfy such property.

Theorem 1 (sufficient condition). *Given the n_{th} input sample X^n , output $S^n = g(X^n, W)$ is a firing rate vector in the synaptic sampling networks and W represents adjustable parameters with internal noise dW . In the classification setting, the output S^n can be interpreted as the scores or probabilities of each class, or as the recognized class label of input sample X^n . A loss function $\phi(H^n, g(X^n, W))$ measures the discrepancy between the desired output for input X^n , and the output $S^n = g(X^n, W)$ computed by the networks. One objective function $f(W) = \mathbb{E}(\phi(H, g(X, W)))$ is average loss function $\phi(H^n, g(X^n, W))$ over a set of labeled examples $\{(X^1, H^1), \dots, (X^N, H^N)\}$. The supervised learning problem is to find the local minimum W of objective function $f(W)$, if the internal noise dW satisfies Equation (4), function f is strict saddle.*

$$dW = \mathcal{N}(0, N\alpha e^w)dt \quad (4)$$

Proof sketch of Theorem. There are mainly two difficulties in the proof of Theorem 1: how to transfer noise distribution to the computable function and how to prove $\lambda_{\max}(\nabla^2 f(\theta)) > 0$ according to the definition of strict saddle condition in section 2. For the first difficulty, due to the Gauss property $p\{|x - \mu| < \sigma\} = 0.6826$ and $\frac{w_{ki}e^{w_{ki}}}{\sqrt{e^{w_{ki}}}}|_{w_{ki} \rightarrow 0} = 0$, $\pm N\alpha w_{ki}e^{w_{ki}}$ represents the general characteristic of noise distribution appropriately. Therefore, it is plausible to refer to $(\sum_n \alpha w_{ki}e^{w_{ki}}(S_k(t) - \Theta\{h_k^n\}))dt$ as the noise distribution dW_{ki} in the computation. For the second difficulty, according to the definition of strict saddle condition in section 2, the sufficient condition of strict saddle property is $\lambda_{\max}(\nabla^2 f(\theta)) > 0$. However, it is difficult to compute λ_{\max} directly. In fact, it is convenient to compute a stronger condition, i.e., $\sum \lambda(\nabla^2 f(\theta)) > 0$. According to the equation about trace of $n \times n$ matrix A : $tr(A) = \lambda_1 + \dots + \lambda_n$, we just concentrate on the diagonal elements of the Hessian matrix. For computational convenience, we convert the derivative of θ to w according to the chain rule. We get that $\nabla f(\theta) = 0 \Leftrightarrow \nabla f(w) = 0$ and $\sum_k \sum_i \nabla f(\theta_{ki}) \geq 0 \Leftrightarrow \sum_k \sum_i (\nabla f(w_{ki}))w_{ki}^2 \geq 0$. According to the equation about trace of matrix M : $tr(M) = \sum \lambda$, we get,

$$\sum \lambda(\nabla^2 f(\theta)) = \sum_k \sum_i \nabla^2 f(\theta_{ki}) \quad (5)$$

$$= -\frac{KI}{\sigma^2} + \sum_k \sum_i \frac{1}{\sigma^2}(\theta_{ki} - \mu) + \sum_n \sum_k \sum_i w_{ki}\alpha e^{w_{ki}}(S_k(t) - \Theta\{h_k^n\})$$

It is obvious that $\sum \lambda(\nabla^2 f(\theta))$ consists of three terms: $A = -\frac{KI}{\sigma^2}$, $B = \sum_k \sum_i \frac{1}{\sigma^2}(\theta_{ki} - \mu)$, $C = \sum_n \sum_k \sum_i w_{ki}\alpha e^{w_{ki}}(S_k(t) - \Theta\{h_k^n\})$. We need to prove the following equality.

$$\sum \lambda(\nabla^2 f(\theta)) = A + B + C > 0 \quad (6)$$

The proof is divided into three steps. Note that the first sentence of each step below is the conclusion we want to prove.

- 1) $B \ll C$. Only when the noise $dW_{ki} = \mathcal{N}(0, N\alpha e^{w_{ki}})dt$, we can derive that $B + (x_i^n - \alpha e^{w_{ki}})C$ is a variant of the gradient. According to the zero gradient and STDP learning rule, $\frac{B}{C} \approx 0$, thereby B can be ignored.
- 2) C is positive. $C \approx N(\sum_i w_{ki}x_i - \sum_i w_{label,i}x_i)$ which represents the approximate potential difference of actual and expected neurons. When networks are trapped in saddle points, the neuron that releases spikes is not the one expected. Thus, the potential of actual neurons is higher than expected.
- 3) $A + B + C > 0$. A is a negative constant. When N is greater than a certain value, C is large enough so that $A + B + C > 0$ and the strict saddle property will be satisfied.

The theorem is therefore proven. That is to say, Theorem 1 guarantees noisy synaptic sampling networks satisfy the strict saddle condition, and hence noise will help escape from saddle points in Theorem 6 of the work (Ge et al., 2015). The detailed derivation appears in **Appendix 1 (Supplementary Presentation 1)**. It is worth noting that we found that the important step C represents the positive potential difference of actual and expected neurons, and thus the strict saddle condition can be satisfied as long as C is large enough. The realization of such an important step comes from introducing parameters $\pm N\alpha w_{ki}e^{w_{ki}}$ by proposed noise. In other words, noise helps spiking neural networks build appropriate Hessian construction, and optimization can utilize enriched curvature information of the node in the direction without ever calculating or storing the Hessian itself.

4. BIOLOGICALLY INTERPRETATION FOR PROPOSED NOISE

4.1. Origin From the Free-Energy Principle in Neuroscience

The proposed noise structure is inspired by the free-energy principle. It is generally believed in neuroscience that any adaptive system at equilibrium with its environment must minimize its free energy (Friston, 2010; Joffily and Coricelli, 2013; Apps and Tsakiris, 2014; Colombo and Wright, 2018). Free energy can be expressed as self-information plus a Kullback-Leibler divergence term in Equation (7), where \tilde{s} is a sensation signal (Friston, 2010).

$$F = D(q(\vartheta|\mu) || p(\vartheta|\tilde{s})) - \ln p(\tilde{s}|m) \quad (7)$$

Given the noise signals $\tilde{\varepsilon}$, the Kullback–Leibler divergence is the perceptual difference between the recognition density $q(\vartheta|\mu)$ encoded by internal states μ and the posterior density $p(\vartheta|\tilde{\varepsilon})$ of the causes ϑ . Self-information measures the error between the true and expected sensation. It is formally the negative log-probability of a noise outcome $\tilde{\varepsilon}$ given the generative model m , that is, $-\ln p(\tilde{\varepsilon}|m)$. Equivalently, it is also expressed as the long-term average of the square error between the true and expected sensation. The divergence is always non-negative, and free energy is tightly bounded by surprise.

A number of cognition and perception studies show that brain system implicitly infers the cause (network parameters ϑ) in a Bayesian fashion (Ernst and Banks, 2002; Yuille and Kersten, 2006; Beck et al., 2008), the recognition density $q(\vartheta|\mu)$ approximates the posterior density $p(\vartheta|\tilde{\varepsilon})$. That is to say, Kullback–Leibler divergence approximates zero and free energy becomes surprise. Therefore, minimizing the free energy is also a process to minimize the error $E(\varepsilon - \tilde{\varepsilon})^2$ between the true and expected noise outcome $\tilde{\varepsilon}$. Thereby, we obtain the optimal noise distribution as shown in Theorem 2.

Theorem 2 (free energy principle). Suppose that a function $g(X): \mathbb{R}^I \rightarrow \mathbb{R}^K$ is a spiking-based winner-take-all neural network, given the output variable $z^n = k$, the value of input x_i^n is from a Poisson distribution $\text{POISSON}(x_i^n | \alpha e^{w_{ki}})$ where the mean is determined by the synaptic weight w_{ki} from input neuron i to network neuron k . If there is a noise distribution $p(\varepsilon|\theta)$ in Equation (8), such a self-organizing system can minimize its free energy of noise signals. Further, the optimal noise $\hat{\varepsilon}$ is obtained by the minimum mean square-error estimation $E(\varepsilon|\hat{\theta})$.

$$\varepsilon|\theta \sim \mathcal{N}(0, N\alpha e^{w_{ki}}) \quad (8)$$

The important step is to obtain the probability distribution $p(\varepsilon|\theta)$. By inducing input variables x , the unknown distribution $p(\varepsilon|\theta)$ will become the integration of easy distributions $p(\varepsilon|x)$ and $p(x|\theta)$ in Equation (9). $p(x|\theta)$ is the normal distribution where both the mean and variance are $N\alpha e^{w_{ki}}$. It has been shown in the work by Habenschuss et al. (2013) and Kappel et al. (2015) that in the spiking-based WTA networks, one prominent motif of cortical microcircuits, $p(x|\theta)$ is the integration of N Poisson distribution with the mean $\alpha e^{w_{ki}}$, which can approximate normal distribution. The detailed proof appears in **Appendix 2 (Supplementary Presentation 1)**.

$$p(\varepsilon|\theta) = \int p(\varepsilon|x)p(x|\theta)dx \quad (9)$$

In this section, we illustrate three points. First, the free energy principle helps verify the plausibility of the proposed noise. It is popular to use Brownian motion to describe continuous random fluctuations in spiking activities. In contrast, the standard deviation of our noise is $N\alpha e^w$ while it is constant in traditional Brownian noise, which is an important difference to other similar noise models. According to free energy, only this type of noise, i.e., $\mathcal{N}(0, N\alpha e^w)$ can be derived rather than standard Brownian noise or other forms. Therefore, it is strong evidence for the plausibility of the proposed noise theoretically. Second, the free energy principle improves biological relevance

of our noise. In neuroscience, the free energy principle unifies different aspects of how the brain works, such as attention, synaptic plasticity, and neuronal coding. Satisfying the free energy principle complements evidence of the neurobiological existence of our proposed noise. Third, the origin of our proposed noise should be illustrated, and why we choose such type of noise, and not other types of Brownian noise in the strict-saddle condition, is answered. In fact, we derived the proposed noise initially inspired by the free energy principle. We then found that such noise helps spiking neural networks satisfy the strict-saddle condition.

4.2. Consistency With Biological Observations

Many biological and biochemical stochastic processes affect the efficacy of a synaptic connection. Some are indirectly related, for example, NMDA receptors, PSD-95 in the mammalian postsynaptic density (PSD), which can affect the amplitude of postsynaptic potentials and the efficiency on clustering glutamate receptors (Bhalla and Iyengar, 1999; Gray et al., 2006; Coba et al., 2009; Ribault et al., 2011). Some are directly related, such as the volume of spines at dendrites (Engert and Bonhoeffer, 1999; Matsuzaki et al., 2001; Zhong et al., 2005; Ho et al., 2011). It is popular to use Brownian motion $\mathcal{W}(t)$ to describe such random continuous fluctuations (Tuckwell, 1988; Cateau and Fukai, 2003; Cateau and Reyes, 2006; Nobuaki et al., 2009). Brownian motion $\mathcal{W}(t)$ is utilized in the Langevin equation as Equation (10) shown,

$$\frac{dV(t)}{dt} = \sigma(V(t)) \frac{d\mathcal{W}(t)}{dt} + \mu(V(t)) \quad (10)$$

where $V(t)$ represents a stochastic process with an average change (or drift) $\mu(V)$, and standard deviation $\sigma(V)$, $\mathcal{W}(t)$ represents standard Brownian motion. Nobuaki et al. (2009) applied such a stochastic process in Equation (10) to the volume of spines $V(t)$. They recorded the volumes of many individual spines of CA1 pyramidal neurons in a rat hippocampus. They found “intrinsic volume fluctuations” in the absence of synaptic activity. **Figure 3** shows the corresponding quantitative analysis of fluctuations in spine-head volume in the absence of activity-dependent plasticity. It shows that average change $\mu(V)$ of intrinsic volume fluctuations is zero, and the standard deviation $\sigma(V)$ is roughly proportional to the spine-head volume. It is likely because larger spines have a greater PSD area. Therefore, it will accumulate more AMPA-type glutamate receptors and more synaptic vesicles in the presynaptic terminal (Harris and Stevens, 1989; Nusser et al., 1998; Takumi et al., 1999; Harris et al., 2003; Knott et al., 2006). In our noise structure $\mathcal{N}(0, N\alpha e^w)$, σ is also greater for larger spines. It is consistent with this important observation of spine dynamics, and further, it may be a more plausible model to describe intrinsic fluctuations.

5. EXTENSION TO MULTI-LAYER NOISY SPIKING NEURAL NETWORKS

In this section, we will demonstrate one computational application of our noise model: realization of multi-layer spiking neural network for better representations. These characteristics

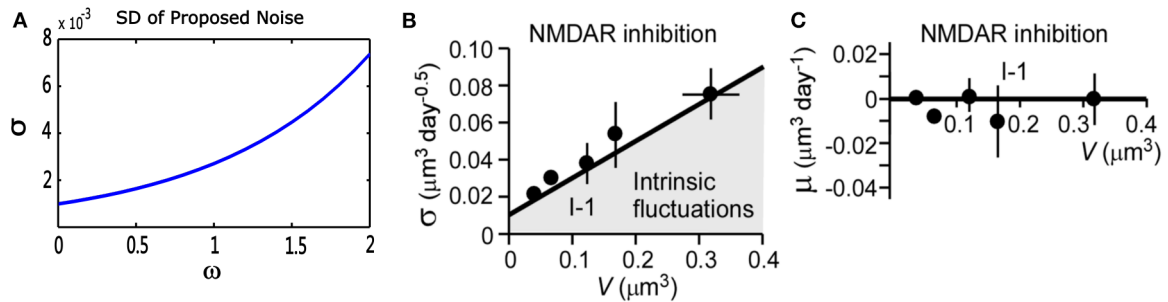


FIGURE 3 | Quantitative analysis of noise from the proposed theory and physiological experiments in the absence of synaptic activity. **(A)** Standard deviation (σ) of proposed theoretical noise $\mathcal{N}(0, N\alpha e^{w_{ji}})$. When synaptic weight changes are relatively small, the standard deviation is roughly proportional to synaptic weights. **(B,C)** Standard deviation (σ) and mean (μ) of fluctuations in spine-head volume in the absence of activity-dependent plasticity in physiological experiments. Standard deviation (σ) is greater for larger spines. Further, (σ) has an approximately proportional relationship with spine-head volume. The mean change (μ) is around zero, which is consistent with that of our noise (cited in Figure 5 of the paper Nobuaki et al., 2009).

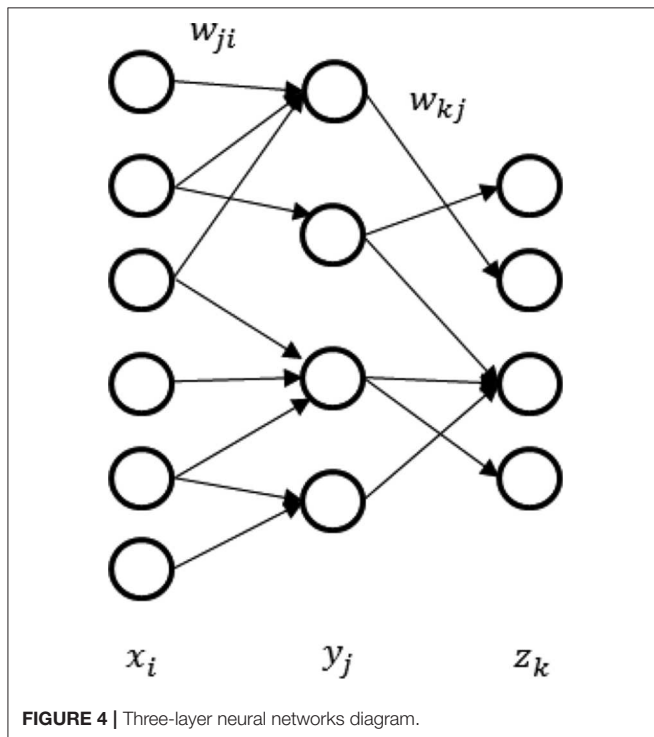


FIGURE 4 | Three-layer neural networks diagram.

will shed new light on machine learning. It is generally believed that deep neural networks can learn representations better than the two-layer network and are more extensively applied in various scenarios (LeCun et al., 2015). It is therefore significant to generalize the depth of noisy spiking neural networks. As an example, we derive a back-propagation algorithm for synaptic sampling on the three-layer network in **Figure 4**. The derivation for deeper networks is similar.

The prior probability remains the same, which reflects the structural constraints and rules. The likelihood function is still the form of cross-entropy, which reflects the class recognition probabilities. The difference is the posterior probability

$p(z^n = k | x^n, w)$ becomes the product of Poisson distributions of both the first and second layers. The likelihood function in **Table 1** becomes,

$$\begin{aligned} \log p_N(J|\theta) &= \sum_{n=1}^N \sum_{k=1}^K \Theta\{h_k^n\} \log p(z^n = k | x^n, w) \\ &= \sum_{n=1}^N \sum_{k=1}^K \Theta\{h_k^n\} \log \frac{\prod_i \sum_j \text{POISSON}(x_i^n | \alpha e^{w_{ji}}) \text{POISSON}(y_j^n | \alpha e^{w_{kj}})}{\sum_k \prod_i \sum_j \text{POISSON}(x_i^n | \alpha e^{w_{ji}}) \text{POISSON}(y_j^n | \alpha e^{w_{kj}})} \end{aligned} \quad (11)$$

The main difficulty is how to get the gradient in the first layer. According to the generalized delta rule, derivative results from the product of two parts: one part represents the change in likelihood function relative to the change of net inputs, and one part reflects the change of net inputs relative to a small change of weight. Thus, we get,

$$\frac{\partial \log p_N(J|\theta)}{\partial w_{ji}} = \frac{\partial \log p_N(J|\theta)}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} \quad (12)$$

where y_j is net inputs for the second layer. For the second factor, as the firing rate of stochastic spike response neurons depends exponentially on the membrane voltage (Jolivet et al., 2006; Mensi, 2011), we derive that,

$$\frac{\partial y_j}{\partial w_{ji}} \approx x_i \quad (13)$$

For the first factor, it can be implemented by propagating gradient information backward through the last layer. According to the chain rule, it can also be written as the product of two parts, as shown in (Equation 14).

$$\frac{\partial \log p_N(J|\theta)}{\partial y_j} = \sum_k \frac{\partial \log p_N(J|\theta)}{\partial w_{kj}} \frac{\partial w_{kj}}{\partial y_j} \quad (14)$$

One part is the gradient of the last layer, and another part is simply the deviation of mean function $E(y_j)$ determined by the synaptic weight w_{kj} from input neuron j to neuron k . In this case, substituting Equations (13) and (14) to Equation (12), we finally get the gradient information of the first layer.

Given the L-layer noisy spiking neural networks, each layer computes a function $X^l = g_l(X^{l-1}, W^l)$, where X^l is the output of the l_{th} layer, X^{l-1} is the input of the l_{th} layer and W^l is the vector of adjustable parameters between the $(l-1)_{th}$ and the l_{th} layer. Note the vector X^1 in the first layer is the input sample. The learning rule for L-layer spiking neural networks can be concluded as follow. The detailed derivation appears in **Appendix 3 (Supplementary Presentation 1)**.

$$d\theta_{kj}^L = b \left(\frac{1}{\sigma^2} (\mu - \theta_{kj}^L) + \sum_{n=1}^N (w_{kj}^L (x_j^{n,L-1} - \alpha e^{w_{kj}^L}) (\Theta\{h_k^n\} - S_k^L(t))) \right) dt + b dW_{kj}^L \quad (15)$$

$$d\theta_{ji}^{l-1} = \beta \alpha \sum_{n=1}^N x_i^{n,l-1} (d\theta_{kj}^l + d\theta_{mj}^l) \quad (2 \leq l \leq L) \quad (16)$$

where $d\theta_{kj}^l$ represents change in parameters corresponding to the neuron k which releases a spike and $d\theta_{mj}^l$ represents change in parameters corresponding to the desired neuron m .

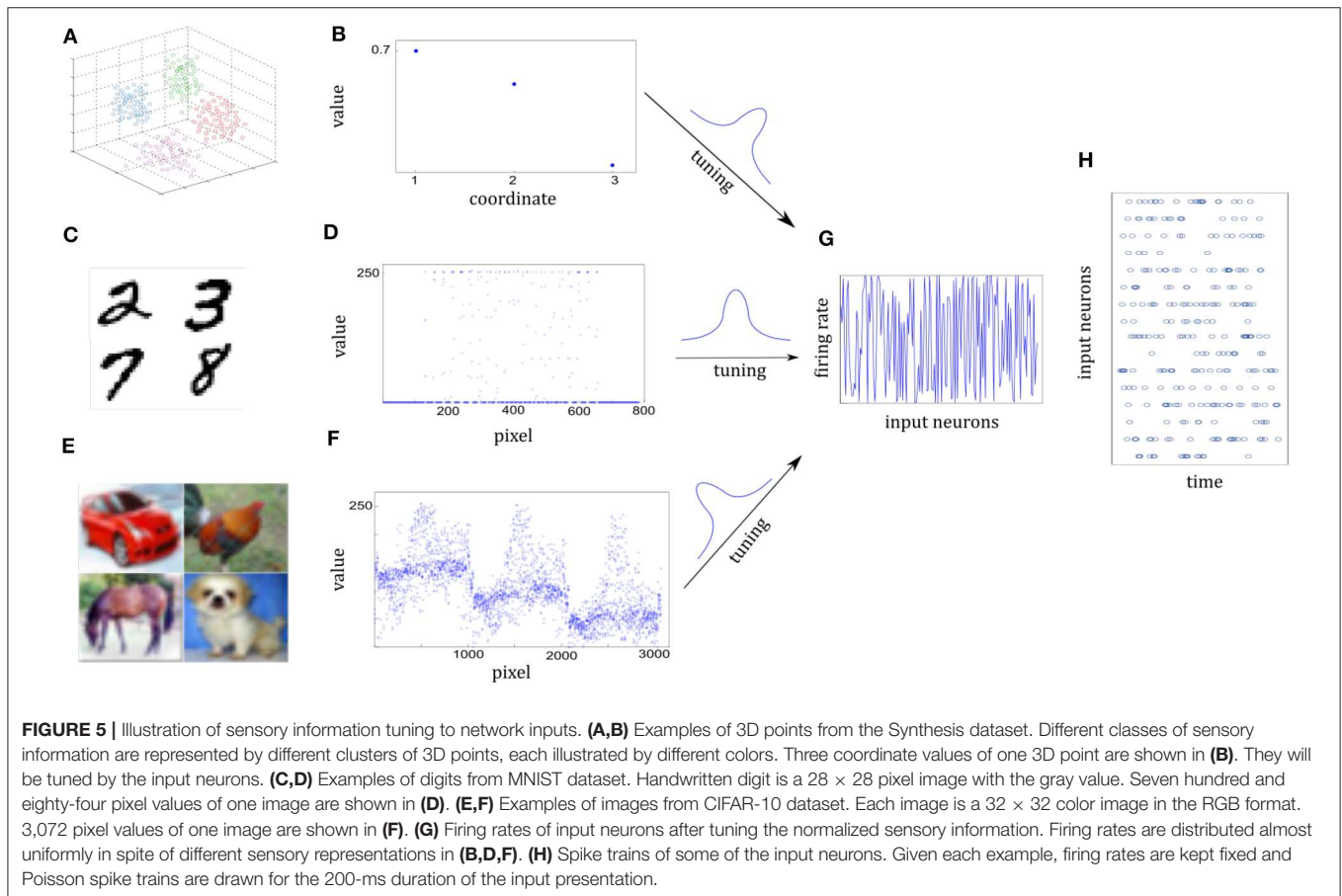
6. SIMULATION RESULTS

We run simulations for synaptic sampling with/without noise applied to 10-categories of classification. We test the proposed model on three datasets from three aspects: (1) application accuracy; (2) neuron spike responses; (3) reduction rates for trapping in saddle points.

6.1. Sensory Environment

6.1.1. Synthesis Dataset

We use a cluster of points in 3D space to represent one sensory experience for visualization. The center of a cluster is the mean of the Gaussian, which is independently drawn from $\mathcal{N}(0.5, 0.2)$. The covariance matrix of the cluster is randomly given by $0.04\mathbb{I} + 0.01\xi$, where \mathbb{I} is the 3-dimensional identity matrix and the element of ξ is randomly drawn from $\mathcal{N}(0, 1)$. **Figure 5A** shows some clusters of points in 3D space. Each cluster represents one class. Different cluster are described by different color. **Figure 5B** shows three-dimensional coordinates. For 10-categories classification, 10 clusters will be generated equally. We generate a sample by randomly selecting one Gaussian cluster and then get a sample position from the corresponding distribution.



6.1.2. MNIST Dataset

It composes of 10 handwritten Arabic numbers from 0 to 9, which has a training set of 60,000 examples, and a test set of 10,000 examples. It is a good compromise between real-world data test and easy preprocessing and formatting. Each example is a 28×28 pixel image with the gray-scale value. In preprocessing the pixel values are normalized to interval $[0, 1]$. Images in **Figure 5C** are drawn from MNIST dataset. **Figure 5D** shows the 784 pixel values of one 28×28 grayscale image.

6.1.3. CIFAR-10 Dataset

It consists of 60,000 color images in 10 classes, which has a training set of 50,000 examples, and a test set of 10,000 examples. The 10 classes are completely mutually exclusive, such as airplane, bird, and cat. Each example is a 32×32 color image in the RGB format. In preprocessing the pixel values are normalized to interval $[0, 1]$. In **Figure 5F**, it consists of three similar parts due to RGB format.

6.1.4. Network Inputs

Both the sample positions in the Synthesis dataset and real-world images from the MNIST or CIFAR-10 dataset are represented by the spatiotemporal spike trains. For the Synthesis dataset, the input layer is 1,000 neurons for one 3D point while for real-world datasets, each pixel of one image is represented by a single input neuron. Input neurons have different Gaussian tuning curves. According to tuning curves, they tune the sample position or normalized pixel values to corresponding firing rates. In addition, the 5 Hz background noise is added. Although raw sensory values of three datasets are differently distributed in **Figures 5B,D,F**, corresponding firing rates are scattered over almost the entire probability space after tuning in **Figure 5G**. As a result, Poisson spike trains of each input neuron are drawn with duration of 200 ms in **Figure 5H**.

6.1.5. Settings

In all simulations, we set $N = 1,000$, $\alpha = e^{-6}$, and $b = 10^{-5}$ or $b = 10^{-4}$. Initial synaptic parameters are drawn from the prior distribution $p_S(\theta)$. We adopt the same configuration about the offset θ_0 and actual weights \hat{w}_{ki} with Kappel et al. (2015)

The purpose of our paper is to propose one appropriate computational hypothesis about whether biologically inherent noise benefits neural systems and how it occurs precisely. To test this hypothesis, we chose the biologically appropriate neural model: synaptic sampling. On the one hand, the inherent noise is described in variables in synaptic sampling and hence it is easy to capture the details of noise biophysics and dynamics. The advantage of noise can be analyzed based on mathematical tractability. On the other hand, synaptic sampling is a biologically appropriate neural system since it simulates some aspects of realistic neural systems, such as neuron topology, neuron type (e.g., excitatory, inhibitory), and Spike Timing Dependent Plasticity (STDP) learning rule, spatial and temporal effect of spike signals. The goal of our paper has been achieved when learning performance is better in the presence of noise than in its absence in the synaptic sampling experiments. Although

it may perform better with further hand-tuning, it is beyond the scope of this paper. As far as we know, we first realize the supervised learning in the synaptic sampling networks. Apart from the stochastic term, the parameter configuration in synaptic sampling without noise is the same as that with noise. Therefore, synaptic sampling without noise is representative of the basic model.

6.2. Verification on the Two-Layer Networks

Through the tuning curves of input neurons, 200 ms spike patterns were communicated to synaptic sampling networks for each sample. According to Equation (3) and spike-based update scheme, the sensory experiences were presented sequentially, and all synapses were updated sequentially. The final predicted label is the neuron which fires most between the 10 output neurons. We repeat the simulation 10 runs and the accuracy is averaged over 10 runs. For the Synthesis dataset, we present 14,400 samples to 1,000 input neurons for 2.4 h. For the MNIST dataset, we present 60,000 samples to 784 input neurons. For the CIFAR-10 dataset, we present 50,000 samples to 3,072 input neurons. As shown in **Figure 6**, in the learning and test phase, the accuracy of synaptic sampling is almost 20% higher than that without noise for the Synthesis dataset. The gap of accuracy with/without noise is around 15% for the CIFAR-10 dataset and around 10% for the MNIST dataset. The accuracy of synaptic sampling without noise in these datasets is, respectively around 80% (MNIST), 39% (CIFAR-10), 60% (Synthesis dataset). It shows the number of bad saddle points in the spiking neural network is relatively small given the MNIST dataset. Therefore, synaptic sampling without noise also obtains satisfactory performance, and adding noise obtains the least increase in accuracy compared to other datasets. On the other hand, the CIFAR-10 dataset is the most challenging among three datasets due to the larger scale and more complex representation as shown in **Figure 5**. It indicates that there will be a large number of bad saddle points making it difficult to achieve a significantly better performance. As a result, the accuracy is only improved from 40% (without noise) to 55% (with noise). For the three datasets, we found that synaptic sampling with/without noise tends to converge at 10,000th iteration. Although the speed of convergence in synaptic sampling with/without noise is similar, synaptic sampling with noise is faster than that without noise, especially in the CIFAR-10 dataset. It is likely because spiking neural networks with noise build more appropriate Hessian construction and utilize enriched curvature information of the node in the direction. As shown in **Figures 6B,D,F**, the standard deviation of synaptic sampling with noise is slightly smaller in spite of additional fluctuations. That is to say, the precision of the optimization performance does not become worse with the effect of noise, which is different from the general idea that noise will lose precision. In addition, the accuracy in the test phase is similar to that in the learning phase. It shows that noise prevents spiking neural networks from overfitting in spite of increasing learning accuracy.

Figure 7 shows the spike trains of 10 readout neurons in the time course of learning. The ordinate displays 10 readout

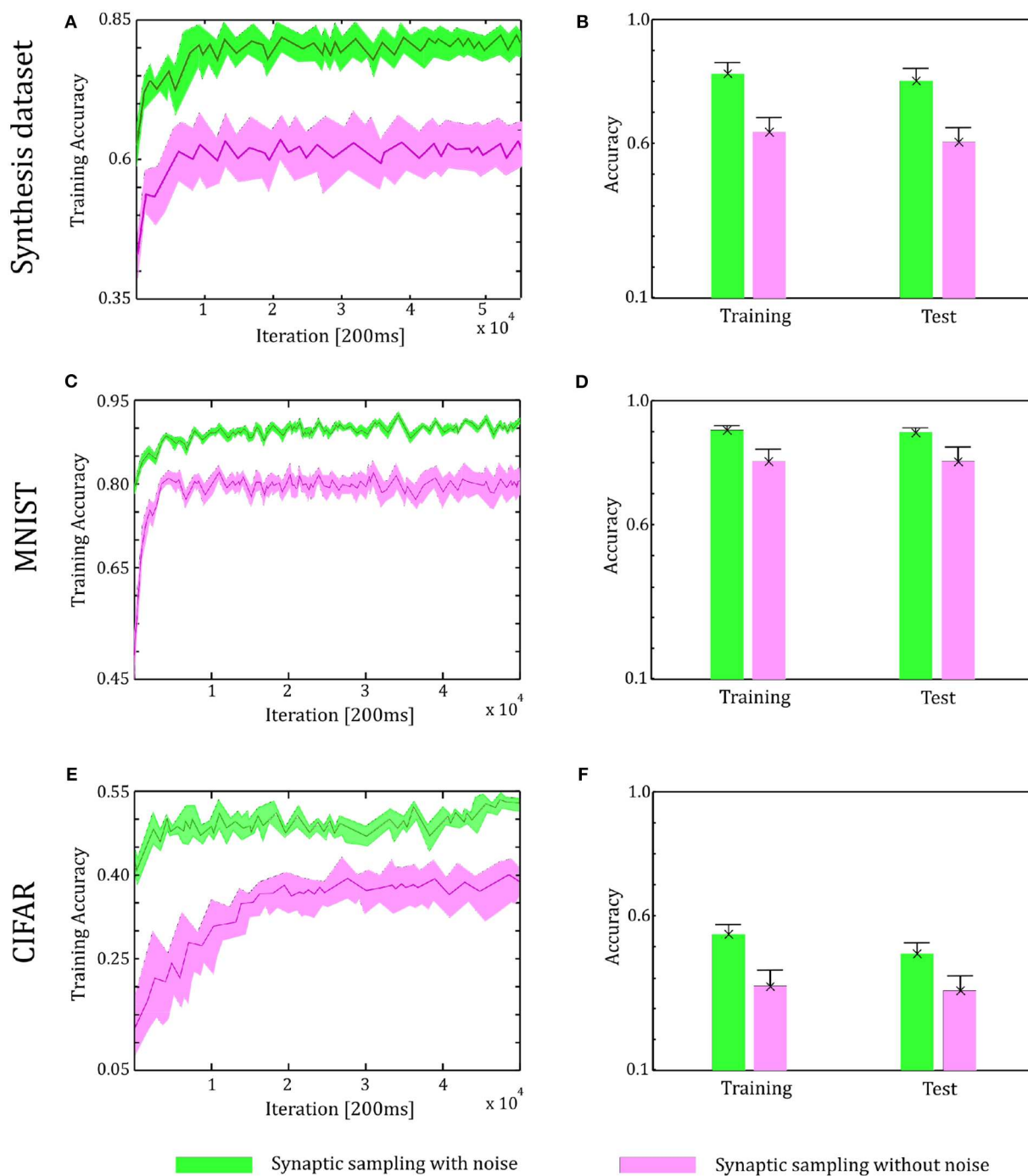


FIGURE 6 | Performance of the two-layer networks with and without internal noise. **(A)** Learning curves of synaptic sampling with/without noise on the training set for the Synthesis dataset. Mean values over 10 runs are shown, shaded area indicates STD. **(B)** Accuracy comparison in the learning and test phase for Synthesis dataset (averaged over 10 runs). Error bars indicate STD. **(C,D)** In the MNIST dataset, the performance with internal noise maintains better than that without noise throughout the whole learning course. **(E,F)** In the CIFAR-10 dataset, the performance with internal noise maintains better than that without noise throughout the whole learning course.

neurons. The abscissa displays time. One point (t, x) represents that at the time t , neuron x releases one spike. In **Figures 7D,E**, it shows the 400-ms learning process from the 57,000th example to

the 57,020th example. The corresponding label is [6, 3, 1, 9, 5, 1, 8, 2, 4, 5, 8, 1, 9, 2, 7, 5, 1, 4, 2, 6]. In the learning process, spikes are scattered almost equally among 10 readout neurons initially and

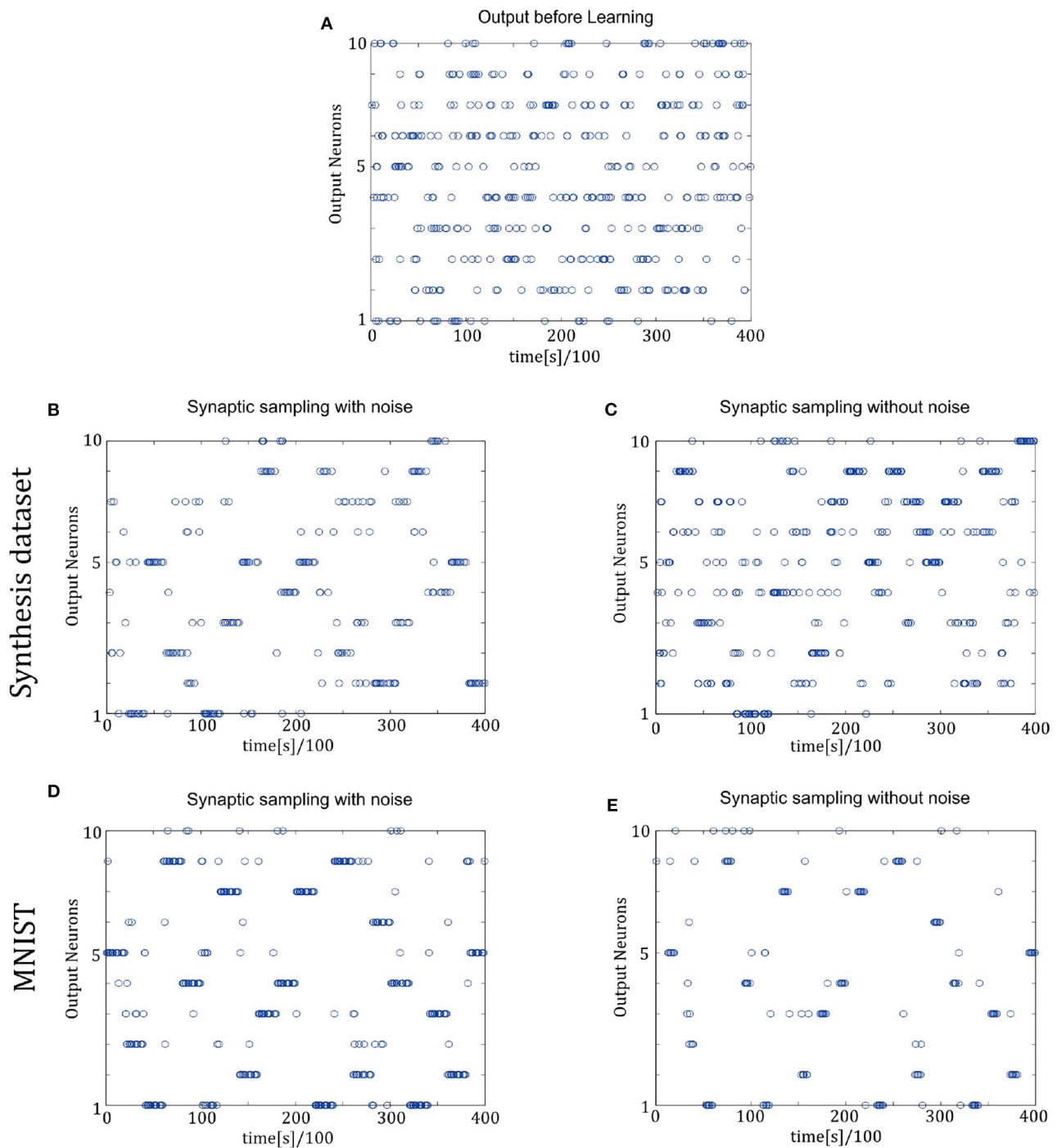


FIGURE 7 | Spiking activity of the 10 output neurons in the two-layer networks with/without internal noise motifs. We present 20 samples during a 4 s epoch for one of the 10 simulations. Ten output neurons in the WTA circuits represent the binary random variables in the supervised classification learning. **(A)** Firing responses of the 10 output neurons before learning. **(B,C)** Firing responses of the output neurons after learning with/without noise for Synthesis dataset. Sparsification of firing of output neurons occurs obviously after learning with noise. **(D,E)** Same for MNIST dataset. The corresponding labels of 20 samples are [6, 3, 1, 9, 5, 1, 8, 2, 4, 5, 8, 1, 9, 2, 7, 5, 1, 4, 2, 6]. The network enters and remains in different network states (indicated by different positions of grouped spikes), corresponding to different predicted class in the supervised learning. The tight match between labels and preference neurons suggests that the generated internal model encodes the MNIST representation efficiently.

hence responses are unspecific to different inputs in **Figure 7A**. However, finally spikes are grouped at the positions from the 6th, 3rd, 1st, 9th, 5th, 1st, 8th, 2nd, 4th, 5th, 8th, 1st, 9th, 2nd, 7th, 5th, 1st, 4th, 2nd, 6th neurons in **Figure 7D**, which is the same as label order. Therefore, responses have become preferences for different inputs. Synaptic sampling with noise obtains the best learning results of this task, and the corresponding spike responses have the most obvious preference pattern. In **Figure 7E**, less spikes are grouped at the positions which is the same as label order. Because synaptic sampling without noise learns this task less accurately. In **Figure 7C**, preference pattern is not very obvious since synaptic sampling without noise cannot learn this task accurately for Synthesis dataset.

Here, we need to discuss one similar work for additional verification of noise effects. Based on synaptic sampling, Kappel et al. (2018) presented a framework to maintain the stable computational power in spite of stochastic spine dynamics. They proposed that the functional role of noise is the compensation for network perturbations and hence noise can help network maintain the stable computational power. They also conducted the experiments about different temperature on learning performance, where the strength of stochasticity can be scaled by the temperature. The results show that good performance was achieved for a range of temperature values, and too low (such as without noise) or too high temperature impaired learning, as shown in the **Figures 5D,G** of their paper. To some extent, it is of evidence that synaptic sampling with noise indeed obtains better performance. However, the difference with our work is outlined as three points.

First, the type of noise is different. They use the standard Brownian noise and the strength of stochasticity should be scaled by the temperature. Therefore, it is an issue to adjust the strength of noise. In contrast, the strength of our noise can be adapted automatically based on the synaptic weight. Second, the perspective from why the noise works is different. They mainly focus on the realization of stable computational function instead of noise mechanism, and therefore provide a short analogy using simulated annealing. However, we analyze the noise theoretically from the view that noise helps optimization escape from saddle points. Third, the application is different. Their model is in the context of reinforcement learning, i.e., lever movement, while our model is in the context of supervised learning, i.e., classification.

6.3. Verification on the Three-Layer Networks

In the simulations of three-layer networks, we present 43,200 samples to networks for the Synthesis dataset and 60,000 samples for the MNIST dataset. Other configurations are the same as two-layer networks. The number of neurons in the hidden layer is 500. We repeat the simulation 10 runs and the accuracy is averaged over 10 runs. As shown in **Figure 8**, in the learning and test phase, the accuracy of synaptic sampling is almost 30% higher than that without noise for the Synthesis dataset and around 15% for the MNIST dataset. Compared with the two-layer network in **Figure 6**, when the number of layers increase, the performance becomes better, i.e., the accuracy is improved

from 80% in two-layer networks to 88% in three-layer network for the Synthesis dataset. It is because deeper networks have the potential for more complex representation. However, the accuracy of synaptic sampling without noise has not improved in spite of the increasing number of layers as the number of saddle points increases exponentially. As shown in **Figures 8B,D**, likely, the standard deviation of synaptic sampling with noise is slightly smaller in spite of additional fluctuations on the three-layer networks. Therefore, noise improving optimization without losing precision can also be applied to three-layer spiking neural networks. In addition, the accuracy in the test phase is similar to that in the learning phase. It shows that noise prevents spiking neural networks from overfitting, regardless of the increasing number of layers. In addition, the speed of convergence in synaptic sampling slows down in three-layer networks, especially without noise, compared to **Figures 6C, 8C**. The increasing number of layers leads to the increasing number of saddle points and hence it is more difficult for networks to escape from saddle points without noise.

In **Figure 9**, the spike responses are similar to that of the two-layer networks. For the Synthesis dataset, initial responses are unspecific to different inputs. After learning, synaptic sampling with noise learns the best results of this task, and the corresponding spike responses have the most obvious preference pattern. Synaptic sampling without noise cannot learn this task accurately, and the preference pattern is not very obvious in **Figure 8C**. For the MNIST dataset, **Figures 8D,E** show the 400-ms learning process from the 57,000th example to the 57,020th example. The corresponding labels are [6, 3, 1, 9, 5, 1, 8, 2, 4, 5, 8, 1, 9, 2, 7, 5, 1, 4, 2, 6]. After learning, more spikes are grouped at the position as indicated by the label number, compared to synaptic sampling without noise. Therefore, synaptic sampling learns the task better and spike responses have more obvious preference pattern.

6.4. Verification for Escaping From the Saddle Points

We try to clarify that the gain in performance in the experiments is due to overcoming the saddle point problem. Although it is difficult to calculate the total number of saddle points and strict-saddle points, it is easy to calculate the total number of strict-saddle points which satisfies Equation (6) given the inputs and network weights. Note that strict-saddle points which satisfies Equation (6) is the subset of strict-saddle points, and hence it is plausible to test the existence of escaping from the saddle points with noise. In Equation (5), given different inputs, the Hessian property of a single weight may be different. Therefore, the main configuration of the strict-saddle point depends on the inputs and network weights. Based on the experiments in section 6.2, we choose the 30 weights at the convergence stage and 1,000 inputs for a total 30,000 samples. If Equation (6) is satisfied given one sample, the corresponding weights belong to a strict-saddle point. In this way, we count the number of strict-saddle points in the spiking neural networks with/without noise, denoted as S_1 and S_2 , respectively. Then, we calculate the reduction rate of strict-saddle points. To some extent, the reduction rate reflects how

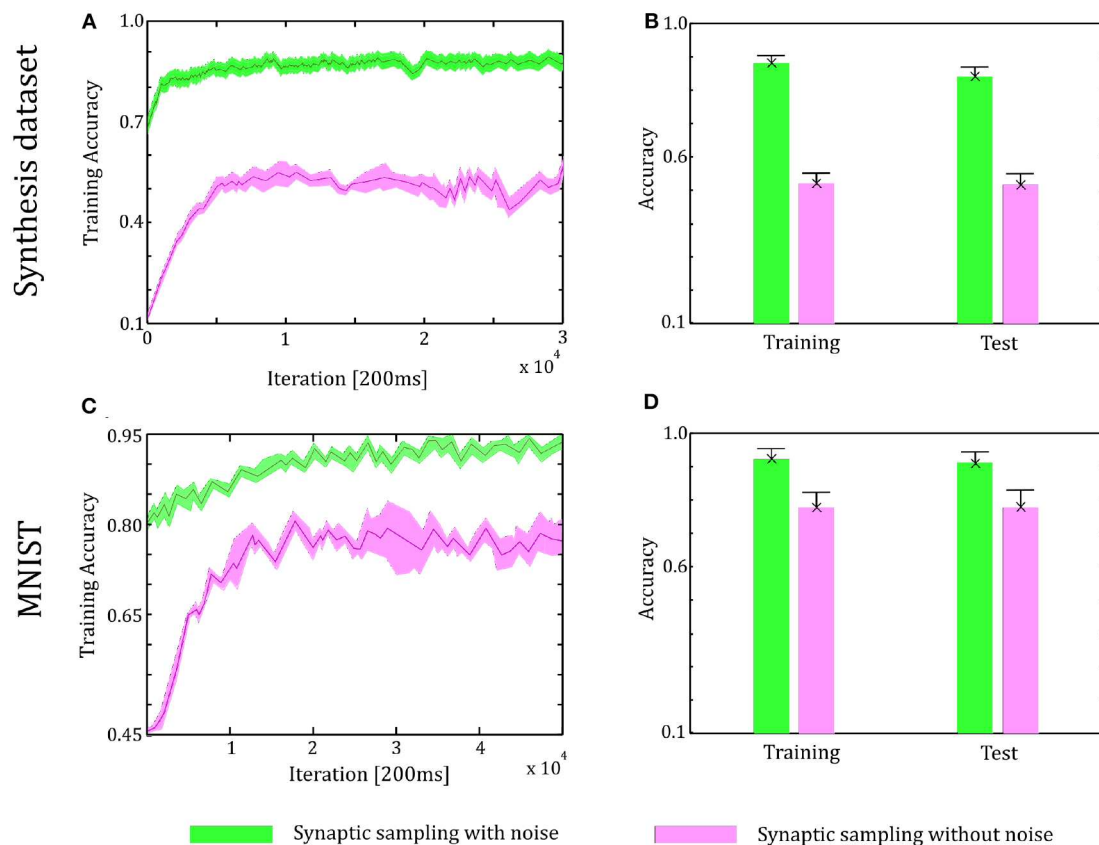


FIGURE 8 | Performance of the three-layer networks with and without internal noise. **(A)** Learning curves of synaptic sampling with/without noise on the training set for Synthesis dataset. Mean values over 10 runs are shown, shaded area indicates STD. **(B)** Accuracy comparison in the learning and test phase for the Synthesis dataset (averaged over 10 runs). Error bars indicate STD. **(C,D)** In the MNIST dataset, the performance with internal noise is better than that without noise throughout the whole learning course.

effectively noise prevents the learning algorithm from trapping in the saddle points. The reduction rate is calculated according to Equation (17).

$$\text{Reduction Rate} = \frac{S_2 - S_1}{S_2} \quad (17)$$

The results are shown in **Table 2**. It obvious that the strict saddle points with noise has been reduced greatly for three datasets, which shows that noise helps optimization escape from many saddle points. In addition, we sort the datasets according to the reduction rate in descending order, that is, the Synthesis dataset > CIFAR-10 > MNIST. According to the accuracy gain in descending order, the list is, Synthesis dataset > CIFAR-10 > MNIST. Therefore, the above two sort lists show that when the reduction rate is greater, performance improves, and noise helps optimization overcome the saddle point problem more efficiently. Such positive correlation indicates that the gain in performance is due to overcoming the saddle point problem as suggested in our manuscript. The number of strict saddle points in the MNIST dataset is the smallest (i.e., 489) and hence synaptic sampling without noise can also obtain the satisfactory performance (~80%). For the CIFAR-10 dataset, although the

reduction rate is not small (68.64%), the number of strict saddle points is still large in contrast (i.e., 6,157) due to the larger scale and more complex representation. Therefore, synaptic sampling with noise only achieves around 54% accuracy.

7. CONCLUSION AND DISCUSSION

In this work, we introduce one biologically plausible noise structure, which is different from the traditional Gaussian noise, and investigate the noise mechanism on the brain computation theoretically. We applied the proposed model to 10-categories classification problem to demonstrate the learning ability of noisy spiking neural networks and compared with networks without noise. Simulation results show that noisy spiking neural networks have higher learning accuracy, and spike responses had a more obvious preference pattern for random spike train inputs. Further, three-layer noisy spiking neural networks have better learning abilities compared with two-layer networks. Our contributions are three fold.

From the perspective of optimization, we propose that one of the essential roles of noise is to improve optimization in the

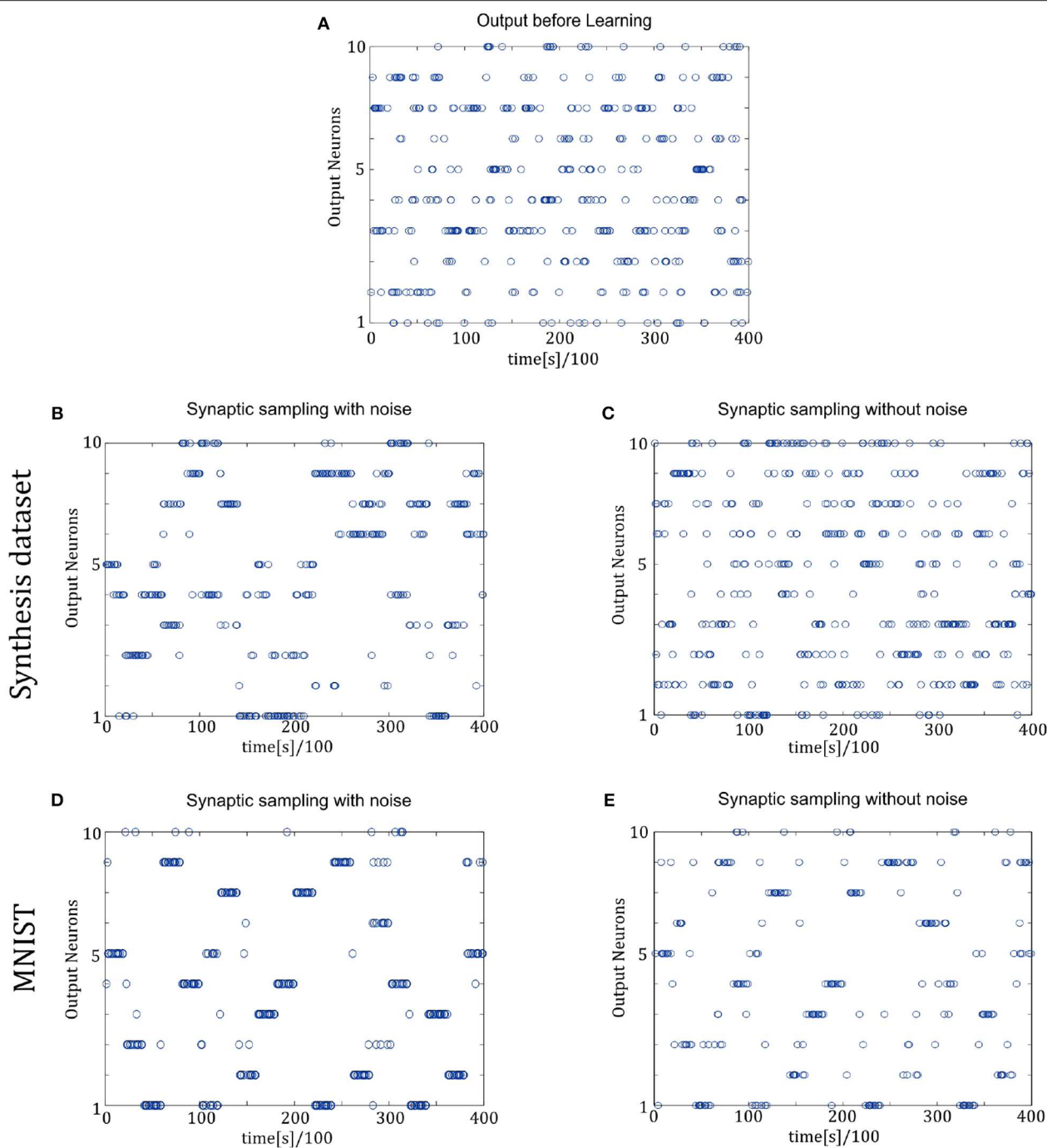


FIGURE 9 | Spiking activity of the 10 output neurons in the three-layer networks with/without internal noise motifs. We present 20 samples during a 4 s epoch for one of the ten simulations. **(A)** Firing response of the 10 output neurons before learning. **(B,C)** Firing response of the output neurons after learning with/without noise for Synthesis dataset. Sparsification of firing of output neurons occurs obviously after learning with noise. **(D,E)** Same for MNIST dataset. The corresponding labels of 20 samples are [6, 3, 1, 9, 5, 1, 8, 2, 4, 5, 8, 1, 9, 2, 7, 5, 1, 4, 2, 6].

brain computations and brain plasticity. Noisy spiking networks for which the synaptic weights affect the noise variance satisfy strict saddle conditions. In other words, proposed noise brings more descent directions and Hessian information of networks

by changing the curvature of the landscape in the network parameter space, especially in the neighborhood near saddle points. In this case, gradient descent with noise will escape from bad saddle points leading to efficient optimization.

TABLE 2 | Reduction results of strict-saddle points and the corresponding gain in accuracy.

Dataset	Number of strict saddle points		Reduction rate (%)	Accuracy (%)		Accuracy gain (%)
	With noise (S_1)	Without noise (S_2)		With noise	Without noise	
Synthesis dataset	711	4,157	82.89	~ 82	~ 60	~ 22
MNIST	489	1,167	58.06	~ 90	~ 80	~ 10
CIFAR-10	6,157	19,632	68.64	~ 54	~ 39	~ 15

From the perspective of biology, we give a theoretical conjecture about the form of noise in the brain. The difference between our noise and traditional Gaussian noise is a positive correlation with a dendritic spine size. There are two biological proofs on such a plausible structure. First, we prove that the probability distribution of proposed noise satisfies the minimum mean square-error estimation based on the free energy principle in neuroscience. Second, it has been observed that larger spines show the most diverse changes in CA1 pyramidal neurons *in vivo* experiments, which is consistent with our noise structure.

From the perspective of the application, our noisy spiking neural networks can be extended to multi-layer networks based on the back-propagation algorithm. Deep learning has excellent abilities in learning complex representations due to its deep network structure. We hope that multi-layer noisy spiking neural networks can serve as a first step toward the realization of more powerful computation.

There are still some related open problems. First, the proposed noise is associated with the number of samples. It is worthwhile to study whether it can be automatically adaptive to some application that is sensitive to the sample size. For example, it has been tested by many studies in machine learning that on-line learning is better in large scale problems, while batch learning is better in small scale problems (Bottou and Bousquet, 2008; Mairal et al., 2009; Hoffman et al., 2010; Lin, 2010; Welling and Teh, 2011; Hardt et al., 2015). We hope the adaptive structure to the sample size in our noise will take effect on the robustness of sample size in artificial intelligence. Another important problem is whether there is a close relationship between proposed noise and similar techniques of adding noise in machine learning such as drop out, on-line learning. In future works, we will try to connect artificial and

biological spiking neural networks further by studying the above two problems.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

YF, ZY, and FC made important contributions to the conception of theory, design, and drafting of the manuscripts. YF designed and performed the simulations. All authors have approved publications in their current form. All authors agree to be responsible for all aspects of the work to ensure proper investigation and resolution of issues related to the accuracy or completeness of any part of the work.

FUNDING

This work is supported in part by the National Natural Science Foundation of China under Grant 61671266, 61836004, in part by the Tsinghua-Guoqiang research program under Grant 2019GQG0006, in part by the National Natural Science Foundation of China under grants 61806011, in part by National Postdoctoral Program for Innovative Talents under grant BX20180005, and in part by China Postdoctoral Science Foundation under Grant 2018M630036.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fnins.2020.00343/full#supplementary-material>

REFERENCES

- Abbott, L. F., and Regehr, W. G. (2004). Synaptic computation. *Nature* 431, 796–803. doi: 10.1038/nature03010
- Apps, M. A. J., and Tsakiris, M. (2014). The free-energy self: a predictive coding account of self-recognition. *Neurosci. Biobehav. Rev.* 41, 85–97. doi: 10.1016/j.neubiorev.2013.01.029
- Beck, J. M., Ma, W. J., Kiani, R., Hanks, T., Churchland, A. K., Roitman, J., et al. (2008). Probabilistic population codes for Bayesian decision making. *Neuron* 60, 1142–1152. doi: 10.1016/j.neuron.2008.09.021
- Bhalla, U. S., and Iyengar, R. (1999). Emergent properties of networks of biological signaling pathways. *Science* 283, 381–387. doi: 10.1126/science.283.5400.381
- Bi, G. Q., and Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Neuroscience* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998
- Bottou, L., and Bousquet, O. (2008). “The tradeoffs of large scale learning,” in *Advances in Neural Information Processing Systems* (Vancouver, BC), 161–168.
- Branco, T., and Staras, K. (2009). The probability of neurotransmitter release: variability and feedback control at single synapses. *Nat. Rev. Neurosci.* 10, 373–383. doi: 10.1038/nrn2634
- Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biol. Cybern.* 95:1. doi: 10.1007/s00422-006-0068-6

- Câteau, H., and Reyes, A. D. (2006). Relation between single neuron and population spiking statistics. *Phys. Rev. Lett.* 96:058101. doi: 10.1103/PhysRevLett.96.058101
- Cateau, H., and Fukai, T. (2003). A stochastic method to predict the consequence of arbitrary forms of spike-timing-dependent plasticity. *Neural Comput.* 15, 597–620. doi: 10.1162/089976603321192095
- Cessac, B. (2010). A view of neural networks as dynamical systems. *Int. J. Bifur. Chaos* 20, 1585–1629. doi: 10.1142/S0218127410026721
- Cessac, B. (2011). A discrete time neural network model with spiking neurons: II: Dynamics with noise. *J. Math. Biol.* 62, 863–900. doi: 10.1007/s00285-010-0358-4
- Coba, M. P., Pocklington, A. J., Collins, M. O., Kopanitsa, M. V., Uren, R. T., Swamy, S., et al. (2009). Neurotransmitters drive combinatorial multistate postsynaptic density networks. *Sci. Signal.* 2:ra19. doi: 10.1126/scisignal.2000102
- Colombo, M., and Wright, C. (2018). First principles in the life sciences: the free-energy principle, organicism, and mechanism. *Synthese* 1–26. doi: 10.1007/s11229-018-01932-w
- Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Mathematics* 11(6 Pt 1), 2475–2485. Available online at: <http://papers.nips.cc/paper/5486-identifying-and-attacking-the-saddle-point-problem-in-high-dimensional-non-convex-optimization.pdf>
- Deneve, S. (2008). Bayesian spiking neurons I: inference. *Neural Comput.* 20, 91–117. doi: 10.1162/neco.2008.20.1.91
- Du, S. S., Jin, C., Lee, J. D., Jordan, M. I., Singh, A., and Poczoz, B. (2017). “Gradient descent can take exponential time to escape saddle points,” in *Advances in Neural Information Processing Systems* (Long Beach, CA), 1067–1077.
- Duarte, A., Ost, G., and Rodríguez, A. A. (2015). Hydrodynamic limit for spatially structured interacting neurons. *J. Stat. Phys.* 161, 1163–1202. doi: 10.1007/s10955-015-1366-y
- Engert, F., and Bonhoeffer, T. (1999). Dendritic spine changes associated with hippocampal long-term synaptic plasticity. *Nature* 399:66. doi: 10.1038/19978
- Ernst, M. O., and Banks, M. S. (2002). Humans integrate visual and haptic information in a statistically optimal fashion. *Nature* 415, 429–433. doi: 10.1038/415429a
- Faisal, A. A., Selen, L. P. J., and Wolpert, D. M. (2008). Noise in the nervous system. *Nat. Rev. Neurosci.* 9, 292–303. doi: 10.1038/nrn2258
- Fellous, J. M., Tiesinga, P. H., Thomas, P. J., and Sejnowski, T. J. (2004). Discovering spike patterns in neuronal responses. *J. Neurosci.* 24, 2989–3001. doi: 10.1523/JNEUROSCI.4649-03.2004
- Friston, K. (2010). The free-energy principle: a unified brain theory? *Nat. Rev. Neurosci.* 11, 127–138. doi: 10.1038/nrn2787
- Fyodorov, Y. V., and Williams, I. (2007). Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity. *J. Stat. Phys.* 129, 1081–1116. doi: 10.1007/s10955-007-9386-x
- Galves, A., and Löcherbach, E. (2013). Infinite systems of interacting chains with memory of variable length—a stochastic model for biological neural nets. *J. Stat. Phys.* 151, 896–921. doi: 10.1007/s10955-013-0733-9
- Galves, A., and Löcherbach, E. (2016). Modeling networks of spiking neurons as interacting processes with memory of variable length. *J. French Stat. Soc.* 157, 17–32.
- Ge, R., Huang, F., Jin, C., and Yuan, Y. (2015). “Escaping from saddle points—online stochastic gradient for tensor decomposition,” in *Conference on Learning Theory* (Paris), 797–842.
- Gerstner, W., and Kistler, W. (2002). *Spiking Neuron Models: An Introduction*. New York, NY: Cambridge University Press. doi: 10.1017/CBO9780511815706
- Gray, N. W., Weimer, R. M., Bureau, I., and Svoboda, K. (2006). Rapid redistribution of synaptic PSD-95 in the neocortex *in vivo*. *PLoS Biol.* 4:e370. doi: 10.1371/journal.pbio.0040370
- Habenschuss, S., Puh, H., and Maass, W. (2013). Emergence of optimal decoding of population codes through STDP. *Neural Comput.* 25, 1371–1407. doi: 10.1162/NECO_a_00446
- Hardt, M., Recht, B., and Singer, Y. (2015). Train faster, generalize better: stability of stochastic gradient descent. in *Proceedings of The 33rd International Conference on Machine Learning* (New York, NY), 48, 1225–1234.
- Harris, K. M., Fiala, J. C., and Linnæa, O. (2003). Structural changes at dendritic spine synapses during long-term potentiation. *Philos. Trans. R. Soc. Lond.* 358, 745–748. doi: 10.1098/rstb.2002.1254
- Harris, K. M., and Stevens, J. K. (1989). Dendritic spines of CA 1 pyramidal cells in the rat hippocampus: serial electron microscopy with reference to their biophysical characteristics. *J. Neurosci.* 9, 2982–2997. doi: 10.1523/JNEUROSCI.09-08-02.1989
- Ho, V. M., Lee, J. A., and Martin, K. C. (2011). The cell biology of synaptic plasticity. *Science* 334, 623–628. doi: 10.1126/science.1209236
- Hoffman, M. D., Blei, D. M., and Bach, F. R. (2010). “Online learning for latent dirichlet allocation,” in *International Conference on Neural Information Processing Systems* (Vancouver, BC), 856–864.
- Holcman, D., Korkotian, E., and Segal, M. (2005). Calcium dynamics in dendritic spines, modeling and experiments. *Cell Calc.* 37, 467–475. doi: 10.1016/j.ceca.2005.01.015
- Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. (2017). “How to escape saddle points efficiently,” in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70 (Sydney, NSW), 1724–1732.
- Joffily, M., and Coricelli, G. (2013). Emotional valence and the free-energy principle. *PLoS Comput. Biol.* 9:e1003094. doi: 10.1371/journal.pcbi.1003094
- Jolivet, R., Rauch, A., Lüscher, H.-R., and Gerstner, W. (2006). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Comput. Neurosci.* 21, 35–49. doi: 10.1007/s10827-006-7074-5
- Kappel, D., Habenschuss, S., Legenstein, R., and Maass, W. (2015). Network plasticity as Bayesian inference. *PLoS Comput. Biol.* 11:e1004485. doi: 10.1371/journal.pcbi.1004485
- Kappel, D., Legenstein, R., Habenschuss, S., Hsieh, M., and Maass, W. (2018). A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *Eneuro* 5. doi: 10.1523/ENEURO.0301-17.2018
- Knill, D. C., and Pouget, A. (2004). The Bayesian brain: the role of uncertainty in neural coding and computation. *Trends Neurosci.* 27, 712–719. doi: 10.1016/j.tins.2004.10.007
- Knott, G. W., Holtmaat, A. L., Welker, E., and Svoboda, K. (2006). Spine growth precedes synapse formation in the adult neocortex *in vivo*. *Nat. Neurosci.* 9, 1117–1124. doi: 10.1038/nn1747
- Larremore, D. B., Shew, W. L., Edward, O., Francesco, S., and Restrepo, J. G. (2014). Inhibition causes ceaseless dynamics in networks of excitable nodes. *Phys. Rev. Lett.* 112:138103. doi: 10.1103/PhysRevLett.112.138103
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539
- Lin, X. (2010). Dual averaging method for regularized stochastic learning and online optimization. *J. Mach. Learn. Res.* 11, 2543–2596.
- Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. *Proc. IEEE* 102, 860–880. doi: 10.1109/JPROC.2014.2310593
- Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res.* 11, 19–60. doi: 10.1145/1756006.1756008
- Matsuzaki, M., Ellis-Davies, G. C., Nemoto, T., Miyashita, Y., Iino, M., and Kasai, H. (2001). Dendritic spine geometry is critical for AMPA receptor expression in hippocampal CA1 pyramidal neurons. *Nat. Neurosci.* 4:1086. doi: 10.1038/nn736
- Mensi, S. (2011). “From stochastic nonlinear integrate-and-fire to generalized linear models,” in *International Conference on Neural Information Processing Systems* (Granada), 1377–1385.
- 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
- Mori, T., and Kai, S. (2002). Noise-induced entrainment and stochastic resonance in human brain waves. *Phys. Rev. Lett.* 88:218101. doi: 10.1103/PhysRevLett.88.218101

- 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
- Nobuaki, Y., Masanori, M., Takashi, M., Jun, N., and Haruo, K. (2009). Principles of long-term dynamics of dendritic spines. *J. Neurosci.* 28, 13592–13608. doi: 10.1523/JNEUROSCI.0603-08.2008
- Nusser, Z., Lujan, R., Laube, G., Roberts, J. D., Molnar, E., and Somogyi, P. (1998). Cell type and pathway dependence of synaptic ampa receptor number and variability in the hippocampus. *Neuron* 21, 545–559. doi: 10.1016/S0896-6273(00)80565-6
- Pouget, A., Beck, J. M., Ma, W. J., and Latham, P. E. (2013). Probabilistic brains: knowns and unknowns. *Nat. Neurosci.* 16, 1170–1178. doi: 10.1038/nn.3495
- Ribault, C., Sekimoto, K., and Triller, A. (2011). From the stochasticity of molecular processes to the variability of synaptic transmission. *Nat. Rev. Neurosci.* 12, 375–387. doi: 10.1038/nrn3025
- Sjöström, P. J., Turrigiano, G. G., and Nelson, S. B. (2002). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* 32, 1149–1164. doi: 10.1016/S0896-6273(01)00542-6
- Soula, H., Beslon, G., and Mazet, O. (2004). Spontaneous dynamics of asymmetric random recurrent spiking neural networks. *Neural Comput.* 18, 60–79. doi: 10.1162/089976606774841567
- Takumi, Y., Ramírez-León, V., Laake, P., Rinvik, E., and Ottersen, O. P. (1999). Different modes of expression of AMPA and NMDA receptors in hippocampal synapses. *Nat. Neurosci.* 2:618. doi: 10.1038/10172
- Tuckwell, H. C. (1988). *Introduction to Theoretical Neurobiology*. New York, NY: Cambridge University Press. doi: 10.1017/CBO9780511623202
- Welling, M., and Teh, Y. W. (2011). “Bayesian learning via stochastic gradient langevin dynamics,” in *International Conference on International Conference on Machine Learning* (Bellevue, WA), 681–688.
- Yuille, A., and Kersten, D. (2006). Vision as Bayesian inference: analysis by synthesis? *Trends Cogn. Sci.* 10, 301–308. doi: 10.1016/j.tics.2006.05.002
- Zhong, X., Hugarir, R. L., and Penzes, P. (2005). Activity-dependent dendritic spine structural plasticity is regulated by small GTPase Rap1 and its target AF-6. *Neuron* 48, 605–618. doi: 10.1016/j.neuron.2005.09.027

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.

Copyright © 2020 Fang, Yu and Chen. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks

Alexander Kugele^{1,2*}, Thomas Pfeil², Michael Pfeiffer² and Elisabetta Chicca¹

¹ Faculty of Technology and Center of Cognitive Interaction Technology (CITEC), Bielefeld University, Bielefeld, Germany,

² Bosch Center for Artificial Intelligence, Renningen, Germany

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

Jim Harkin,
Ulster University, United Kingdom
Guoqi Li,
Tsinghua University, China

*Correspondence:

Alexander Kugele
alexander.kugele@de.bosch.com

Specialty section:

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

Received: 14 November 2019

Accepted: 09 April 2020

Published: 05 May 2020

Citation:

Kugele A, Pfeil T, Pfeiffer M and
Chicca E (2020) Efficient Processing
of Spatio-Temporal Data Streams
With Spiking Neural Networks.
Front. Neurosci. 14:439.
doi: 10.3389/fnins.2020.00439

Spiking neural networks (SNNs) are potentially highly efficient models for inference on fully parallel neuromorphic hardware, but existing training methods that convert conventional artificial neural networks (ANNs) into SNNs are unable to exploit these advantages. Although ANN-to-SNN conversion has achieved state-of-the-art accuracy for static image classification tasks, the following subtle but important difference in the way SNNs and ANNs integrate information over time makes the direct application of conversion techniques for sequence processing tasks challenging. Whereas all connections in SNNs have a certain propagation delay larger than zero, ANNs assign different roles to feed-forward connections, which immediately update all neurons within the same time step, and recurrent connections, which have to be rolled out in time and are typically assigned a delay of one time step. Here, we present a novel method to obtain highly accurate SNNs for sequence processing by modifying the ANN training before conversion, such that delays induced by ANN rollouts match the propagation delays in the targeted SNN implementation. Our method builds on the recently introduced framework of streaming rollouts, which aims for fully parallel model execution of ANNs and inherently allows for temporal integration by merging paths of different delays between input and output of the network. The resulting networks achieve state-of-the-art accuracy for multiple event-based benchmark datasets, including N-MNIST, CIFAR10-DVS, N-CARS, and DvsGesture, and through the use of spatio-temporal shortcut connections yield low-latency approximate network responses that improve over time as more of the input sequence is processed. In addition, our converted SNNs are consistently more energy-efficient than their corresponding ANNs.

Keywords: spiking neural networks, sequence processing, efficient inference, neuromorphic computing, event-based vision

1. INTRODUCTION

Spiking neural networks (SNNs) were initially developed as biophysically realistic models of information processing in nervous systems (Rieke et al., 1999; Gerstner et al., 2014), but they are also ideally suited to process data from event-based sensors (Posch et al., 2010; Liu and Delbruck, 2010; Furber et al., 2013; O'Connor et al., 2013; Osswald et al., 2017), and are natively implemented on various neuromorphic computing platforms (Schemmel et al., 2010; Furber et al., 2013; Merolla et al., 2014; Qiao et al., 2015; Martí et al., 2015; Davies et al., 2018). Their sparse

and event-driven mode of computation makes them more energy-efficient and faster compared to conventional artificial neural networks (ANNs), and additionally allows for the use of spatio-temporal spike codes to represent complex relationships between features in the network. These hypothetical advantages can, however, only be completely exploited on hardware that supports *fully parallel model execution*, which means that spiking neurons operate independently from each other and their update is solely based on incoming spikes. This is different from typical ANN execution schemes, which update all neurons in a fixed order determined by the network architecture and at fixed discrete time steps.

The goal of this article is to develop a framework for obtaining SNNs that run fully in parallel and achieve high accuracy, low latency, and high energy-efficiency on sequence processing tasks, in particular classifying streams of events from neuromorphic sensors. Sequence processing seems to be a natural fit for the execution mode of SNNs where every neuron has its own dynamics, but in practice it has proven to be very challenging to exploit this property to train SNNs on temporally varying input data. Even more, current state-of-the-art methods for SNN training are unable to yield competitive accuracies compared to ANNs even in the simpler case of static inputs (Pfeiffer and Pfeil, 2018), albeit the gap has become narrower over the past years due to better training algorithms, such as e.g., variants of backpropagation for SNNs (Lee et al., 2016; Wu et al., 2018; Shrestha and Orchard, 2018; Neftci et al., 2019). However, Deng et al. (2020) argue that SNNs in general are put at a disadvantage in tasks designed for ANNs, such as image classification, because of the information loss incurred during conversion of images to spike trains of finite time window length. SNNs should not be expected to outperform ANNs in terms of accuracy on frame-based tasks, but they may be advantageous in terms of memory and compute costs. SNNs should ideally always be evaluated on event-based datasets, where they are able to outperform ANNs by exploiting the spatio-temporal information encoding of event-streams. Consequently, in this article we use only event-based datasets to evaluate our SNN performance and report memory and compute requirements for our networks, as suggested in Deng et al. (2020).

The currently most successful method for obtaining accurate SNNs is to train an ANN with conventional deep learning methods, and convert the resulting ANN architecture and weights into an equivalent SNN, translating analog neuron activations into proportional firing rates of spiking neurons (Cao et al., 2015; Rueckauer et al., 2017). Conversion methods have achieved the best known SNN accuracies for image classification tasks, such as MNIST, but they rely on the assumption that input patterns do not change for some time. This is required because firing rates in each layer need time to converge to their targets derived from ANN activations. Spikes are allowed to propagate instantaneously between layers of the network, since this speeds up convergence of firing rates in deeper layers, and there is no additional temporal information beyond rates encoded in spike trains.

These assumptions are no longer valid when sequence processing tasks are considered, which require networks capable

of temporal integration. Temporal integration means that information from different times of the input has to be integrated at a single point in time at the output of the network. In a multi-layer network this means that the network architecture as well as the propagation delays between layers become crucial to control not just what features of the input are computed, but also when information computed in other layers can be used to update the feature computation. Temporal integration is achieved with recurrent or *temporal skip connections*, which not only skip layers in depth-direction of the network, but also bridge time like recurrent connections. Since temporal skip connections, in contrast to recurrent connections, serve as shortcuts in time, and hence, reduce the latency of early approximate network responses, we omit recurrent connections in the following.

Our goal is to obtain SNNs for model-parallel execution on actual neuromorphic systems, which requires assigning non-zero delays to all connections in the network. However, current ANN-to-SNN conversion methods are unable to deal with the case of time-varying inputs or with temporal skip connections with different propagation delays. The main contribution of this paper is to close these gaps by unifying ANN-to-SNN conversion with the recently introduced concept of streaming rollouts (Fischer et al., 2018), thereby greatly extending the applicability of SNN training methods to novel and important classes of applications. Since the inference graph of an SNN determines the way temporal information is being processed, its temporal structure needs already to be taken into account during ANN training (see Section 2.2 for details). In other words, it has to be ensured that information from all required parts of the input sequence and the resulting activations of intermediate layers arrives at the right time at the output neurons both during ANN training and after conversion to SNNs. With this novel method for rolling out and training ANNs before conversion to SNNs we obtain SNNs that efficiently and accurately solve sequence processing tasks, and yield approximate responses as early as possible.

In the following, we describe our methods in detail and show experimental results that emphasize the advantages of our approach for event-based sequence processing tasks.

2. METHODS

In this section, we describe the task of classifying event-based data streams with spiking neural networks (Section 2.1), and present a recipe for obtaining SNNs to process input sequences on neuromorphic hardware. First, we define the targeted inference graph of SNNs (Section 2.2) and, then, describe how to train (Section 2.3) and convert (Section 2.4) corresponding artificial neural networks (ANNs). Last, we describe how we estimate the energy-efficiency of both approaches in Section 2.5.

2.1. Classification With Spiking Neural Networks

We study the task of training an SNN that processes a given input spike sequence \mathbf{S}_{in} into a discrete target output $y \in \{1, \dots, C\}$, where C is the number of available classes. The input \mathbf{S}_{in} is a multi-dimensional spike sequence of dimensionality M , where

$S_{\text{in}}^{(i)} = (t_{\text{in},1}^{(i)}, \dots, t_{\text{in},n(i)}^{(i)})$ defines the spike times of neuron $i \in \{1, \dots, M\}$, and $n(i) \geq 0$ is the number of spikes generated by input neuron i in the input sequence. We define $T_{\text{max}}(S_{\text{in}}) = \max_{i=1, \dots, M} t_{\text{in},n(i)}^{(i)}$ as the length of the complete sequence, i.e., the time of the final input spike to any input neuron, and we denote by $S[t_0, t_1]$ the partial spike train that includes all spikes of S between t_0 and t_1 . We introduce the shortcut $S[t] = S[0, t]$ for all spikes up to time t . The output vector $y(t)$ is computed from the spike trains $S_{\text{out}}[t]$ of a defined output layer of the network after seeing all spikes up to time t , and can be computed in various ways, e.g., by applying the soft max function to the spike counts of all output neurons.

In our experiments the spiking neurons are simple non-leaky Integrate & Fire (IF) neurons without refractory period, as described in Rueckauer et al. (2017). Every neuron i is characterized by its membrane potential $V_i(t)$, which is updated whenever the neuron receives an input spike from another neuron j . In this case we update $V_i(t) \leftarrow V_i(t) + w_{ij}$. If $V_i(t)$ exceeds a threshold voltage V_{th} then the neuron sends out a spike and resets its membrane potential by the following subtraction: $V_i(t) \leftarrow V_i(t) - V_{\text{th}}$. Rueckauer et al. (2017) analytically show how IF neurons can approximate ANN activations with spike rates. It is possible to use alternative neuron models, e.g., leaky integrate-and-fire, but to date no practical benefits have been demonstrated that would warrant their additional analytical and computational complexity. Hence, we consider only IF models in this paper.

2.2. Sequence Processing With Streaming Rollouts

The architecture of the neural network is described by a directed *network graph*, in which nodes correspond to layers of a neural network, and edges represent dependencies between the layers. The goal is to train a network for *sequence processing*, which means the output t at any time depends on the entire input sequence $S_{\text{in}}[t]$ or at least a *spatio-temporal receptive field* $S_{\text{in}}[t - \tau, t]$ of duration τ . The network needs to be capable of *temporal integration*, i.e., information about the input in the relevant spatio-temporal receptive field must remain present in some nodes, and must be continuously combined with new incoming information. Temporal integration requires a network graph that includes either recurrent or temporal skip connections, as discussed next.

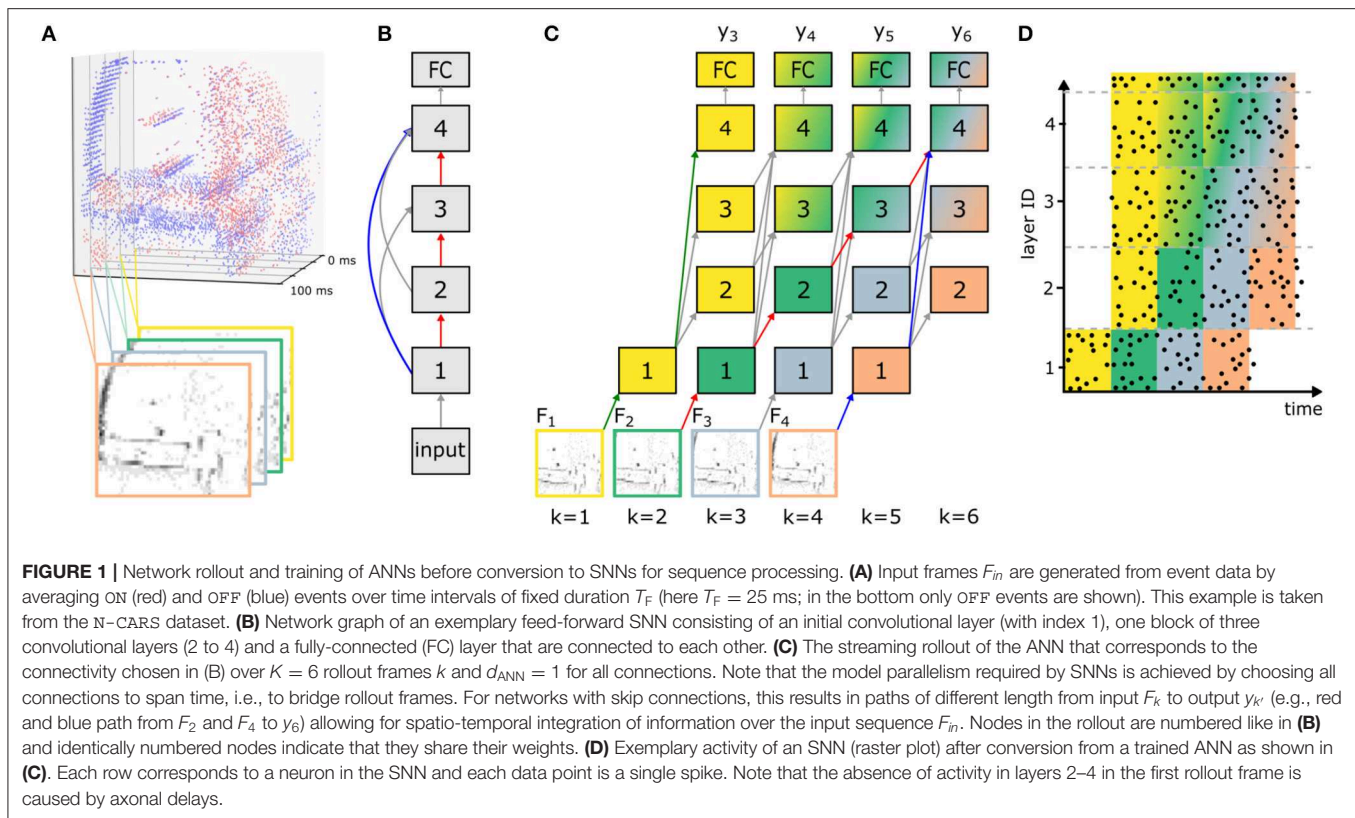
In the setting of an ANN processing an input sequence, a network graph can be rolled out in time in multiple ways (Fischer et al., 2018). The usual convention of *sequential rollouts* is to assume a delay of one time step for recurrent edges, whereas all other edges in the feed-forward direction from input to output are assumed to transport information instantaneously without delay. A mechanism similar to sequential rollouts, although on the granularity of SNN simulation time steps was proposed by Wu et al. (2018) to train SNNs with backpropagation, which allows treating the spatial and temporal domain separately for backpropagation. However, this notion of sequential rollouts is in contrast to the fully parallel execution mode of SNNs, in which all neurons can update their states simultaneously,

but information cannot be instantaneously propagated between neurons. Converting an ANN trained with sequential rollouts into an SNN can therefore lead to a mismatch in the way information is being processed over time.

Fischer et al. (2018) proposed an alternative rollout mechanism called *streaming rollout*, in which all edges transport information to the next *rollout frame*. We define a rollout frame as the state of all neurons at a given time point after applying all instantaneous updates within the same frame, as well as updates from delayed connections from the previous time step(s). The streaming rollout is equivalent to introducing an axonal delay d_{ANN} of at least one rollout frame to all connections. Each neuron's next state can then be computed exclusively from values computed in the previous rollout frame, which allows fully parallel updates within one rollout frame. In previous conversion approaches, the time to reach good approximations scales with the network depth, because the spiking activity in any layer first needs to converge to a good-enough approximation of ANN activations before the next layer is able to generate precise approximations. This is resolved by the streaming rollout, as all layers approximate the activations in parallel, thereby decoupling the depth from the integration time. We limit our analysis in this paper to a delay of one rollout frame.

Skip connections under streaming rollout translate to *temporal skip connections*, which do not only skip layers in the depth-direction of the network, but also span time. Furthermore, temporal skip connections give rise to early approximate results, and the earliest response is determined by the shortest path between input and output in the network graph. Initial predictions are less accurate, because only a shallower network is used for classification, but getting an early guess is desirable for many tasks that require real-time decisions. Note that the scenario we investigate here is different from the typical sequence processing framework, e.g., NLP or speech, where reasonable accuracy can only be obtained after seeing most of the input. The accuracy improves over time as more frames of the input sequence are processed, and as deeper layers of the network begin to contribute to the prediction.

Figure 1 illustrates how streaming rollouts achieve temporal integration for an exemplary network graph (**Figure 1B**) with $N_l = 4$ convolutions and a fully-connected layer (number of blocks $N_b = 1$). The temporal shortcuts with $d_{\text{ANN}} = 1$ allow for temporal integration over multiple frames in the input sequence. This is illustrated in **Figure 1C** by assigning a color to each of the four processed input frames F_1, \dots, F_4 , and the mixing of colors indicates which frames provide information to which layer at every time step. For example, the skip connection from layer 1 to 4 causes early activity in the output layer already at $k = 3$, although with reduced accuracy. Multiple paths of different lengths connect the input to the output, with a shortest path (shown in blue) of length 2, and the longest path (in red) of length 4. The difference between the length of the longest and the shortest path determines the size of the spatio-temporal receptive field. In our example, the size of the receptive field is $\tau = 3$ input frames and, hence, the output in **Figure 1C** is shown as a mix of up to three colors. In **Figure 1C** at $k = 6$, input data from $k = 2$ and 4 arrive at the same time at the



output layer (via the red and blue path, respectively) and can, hence, be jointly used for prediction. In addition, the shortest path between inputs and outputs of such a network (green path) defines the latency, at which a first approximate prediction can be made. Long paths (red path), i.e., deeper networks, allow for better accuracy at the cost of higher computational effort. In addition, the overall energy-efficiency is further increased by regularizing all activations with the L2-norm in order to achieve smaller activations and therefore reduce the number of operations necessary to reach an accuracy level close to the maximum possible. Note that in streaming rollouts newly acquired input frames are immediately processed and fused with pre-processed information from previous inputs to refine the output of the network. Since the computation of all layers in a rollout frame only depends on the outputs of the layers in the last rollout frame, outputs can be computed frame by frame. This is in contrast to other methods for sequence processing, for which multiple input frames are required at once to compute the output of the network (e.g., van den Oord et al., 2016a).

2.3. Training of Artificial Neural Networks

In all our experiments, the network graphs follow the DenseNet architecture (Huang et al., 2017) due to two main reasons. First, DenseNets are established network models and achieve competitive results across various applications (Zhu and Newsam, 2017; Huang et al., 2018; Zhang et al., 2018). Second, the dense connectivity between layers, as described in the following, results in streaming rollouts, in which the output is updated

every time step. In each block of a DenseNet, every layer is connected to all previous layers. The blocks are connected by transition layers that reduce the resolution via pooling. The last layer is composed of global average pooling and a fully-connected layer for classification. Throughout this study, we use network graphs with $N_b = 3$ blocks, all other hyperparameters and a full schematic of a two-block DenseNet can be found in **Appendix A**.

For the streaming rollout of the above network graph, the temporal window τ is limited by the depth of the network $D = N_l N_b + 1$. This explicit restriction to a finite temporal window allows choosing a network architecture that matches the temporal scale of the specific problem at hand. Furthermore, the latency from first input to output in streamingly rolled out DenseNets is N_b rollout frames, which is typically shorter than the latency of D for recurrent networks that utilize all D layers for each prediction. These temporal skip connections in streamingly rolled out DenseNets allow for fast approximate predictions that are refined over time. For our datasets, we saw an increase in our accuracy when replacing regular dropout with spatial dropout (Tompson et al., 2015) and using convolutions with weight kernels of spatial size 3×3 instead of 1×1 for the transition layers (for further hyperparameters, see **Appendix A**).

For the training of ANNs with streaming rollouts, event-based input sequences first need to be converted into sequences F_{in} of N so-called *input frames* F_k (e.g., see **Figures 1A,C**). The input spike sequences S_{in} are divided into N equally sized time intervals of length $T_F = T(S_{in})/N$, and for each interval we compute the sum of all spikes, which is used as the input to the ANN.

Since event-based vision sensors distinguish between ON and OFF events, we compute two separate channels per input frame.

For a given sequence of input frames F_{in} we use streaming rollouts to compute the activations of all ANN units over time, and apply backpropagation-through-time (Werbos, 1990) to train the weights of the network. To consider all N input frames F_k in F_{in} with $k \in \{1, \dots, N\}$ with the shortest possible rollout, the last network output y_k of the rollout is connected to the last input frame F_N , via the shortest path l_s . This results in rollouts with $K = N + l_s$ rollout frames and as many outputs y_k (with $k \in \{l_s, \dots, K\}$) as inputs (for an example, see **Figure 1C**). For every dataset, the number of rollout frames K is determined by τ , the length of the temporal window. If the number of input frames N is smaller than τ , these input frames are evenly distributed over the available τ input slots of the fixed rollouts.

The optimization objective is to minimize the categorical cross entropy L over all predictions y_k of the network outputs,

$$L = \sum_{k=l_s}^K -a_k \hat{y}_k \log(y_k) \quad (1)$$

where \hat{y}_k are the one-hot class labels and a_k are factors to trade off between early and late accuracy. These factors will be discussed in detail in Section 3.2. As we are considering classification problems, the target class label is the same for each output, i.e., $\hat{y}_k = \hat{y} \forall k$. Observe that $\hat{y}_k \log(y_k)$ is a scalar product and since \hat{y}_k is a one-hot vector, only one term is non-zero. In all layers, we apply weight decay as regularization, and activation decay for increased sparsity. For parameterizations and further details, see **Appendix A**.

2.4. ANN-to-SNN Conversion

After training the streaming rollout of the ANN, the architecture and weights of the ANN are translated into an equivalent SNN for energy-efficient inference. We closely follow the conversion method described by Rueckauer et al. (2017), who proved that, under the assumption of ReLU activations and IF neurons, the firing rate r_i of a spiking neuron i becomes proportional to the activation a_i of the corresponding neuron i in the ANN. Hence, for the same input, the output firing rates of the SNN approximate the ANN output activations, and the approximation error decreases with simulation time of the SNN. In order to speed up this approximation, the authors proposed a weight normalization scheme to fully use the dynamic range of the spiking neurons determined by their maximum firing rate.

In this article we go beyond the mechanisms described Rueckauer et al. (2017), and apply ANN-to-SNN conversion to a network rolled out in time using streaming rollouts (see section 2.2 and **Figure 1C**), thereby allowing to address sequence processing tasks. Two levels of temporal integration have to be considered for the SNN: First, for every rollout frame, ANN activations are approximated by firing rates, which happens in the time interval defined by T_F . Therefore, we have to set the duration of a rollout frame long enough for firing rates to converge to their target rates. Second, skip connections in streaming rollouts allow temporal integration of information. For example, the red and

blue path in **Figure 1C** are arriving at the same time at layer 4 at k , because each connection has delay $d_{ANN} = 1$. Consequently, axonal delays of connections in SNNs have to be set such that information propagates through the network as predefined by the rollout of the ANN. If ANN activations in each rollout frame are approximated by n_{sf} simulation steps in SNNs, the delay in SNNs has to satisfy $d = n_{sf} \cdot d_{ANN} = n_{sf}$. Additionally, to prevent neurons from being inactive for too long after receiving a sustained negative input during one rollout frame we use a lower bound on the membrane potential. It is expected that the output rate of a neuron changes smoothly with its input rate. Assuming that the input changes slowly over time, the membrane voltage stored at the end of one rollout frame will be a good initialization for the next rollout frame. In addition, the time until the rate approximation is sufficiently good decreases for each additional rollout frame. The limitation is the resolution with which the rates have to be approximated. A particular advantage of using skip connections is that the time required for information to propagate from input to output is determined by the shortest path l_s . This rate of change from input to output is usually higher than for networks using recurrent instead of skip connections, since for these the shortest path equals to the full depth of the network ($l_s = D$), i.e., information needs to propagate through all layers.

Classification outputs in the final layer of the spiking network are computed as $y(t) = \arg \max(\sum_{t'=t-T_F}^t \mathbf{S}_{out}[t'])$ by summing all weighted input spikes to each neuron over the time interval T_F and taking the $\arg \max$ of this vector. This allows faster adaptation of predictions, does not need an external stimulus, and handles the case when both output neuron activations are negative.

One important aspect of a classification method is the latency between inputs and outputs, especially in scenarios with critical real-time requirements. A direct approach would be to measure the wall-clock time required to execute the SNNs. However, the execution time of SNNs strongly depends on the used hardware system. In order to disentangle this dependency we introduce the hardware-agnostic measure of simulation steps per frame n_{sf} . In a time-stepped SNN simulator, each frame of a sample is used as input for n_{sf} steps. Then, the actual wall-clock time depends on, first, the throughput of the SNN simulator/emulator f in simulation steps per second and, second, the time needed to accumulate one frame T_F (see **Figure 1**). If a new frame is accumulated while the network is executed and $f n_{sf} \leq T_F$ holds, the system runs in real time.

The core idea of ANN-to-SNN conversion is to achieve a linear mapping between activations of the ANN neurons and spike rates of the SNN neurons. Neurons should not saturate, i.e., the rate after mapping should not exceed one spike per simulation step. Therefore, the activations have to be rescaled, which can be achieved by rescaling weights and biases in each layer by a scalar factor. The employed robust scheme scales the parameters of each layer by a predefined percentile of the training set activations, as described in Rueckauer et al. (2017). Rescaling by a percentile of the activations instead of the maximum activation leads to some neurons saturating (they should spike more than once per simulation step), but increases the overall activity in the network, leading to faster propagation

of information and therefore reduces the latency between input and output. Additionally, the authors of Rueckauer et al. (2017) see an increase in accuracy when choosing a percentile as scaling factor instead of the maximum activation. It should be noted that this method is not dependent on the layers used, considers also concatenations of layers and only needs one forward pass to rescale all layers. For our approach, we have to consider that activations change over time. We calculate the percentile over all activations over time, but still for each layer separately. In contrast to the original work, we also rescale the weights of Average Pooling layers by a percentile of the activations and observe an increase in top accuracy.

2.5. Energy-Efficiency and Number of Operations

In order to compare the energy-efficiency of ANNs and SNNs we use the same metric as in Rueckauer et al. (2017), i.e., we measure the average number of operations over all samples in the used dataset split during inference. The number of operations is calculated differently for ANNs and SNNs, due to the difference in their neuron models. As discussed by e.g., Thakur et al. (2018) and Pfeiffer and Pfeil (2018), many different neuron models exist for SNNs depending on the desired biological plausibility and complexity. In this study, we use the IF neuron model as described in Section 2.1 to match the method for ANN-to-SNN conversion introduced by Rueckauer et al. (2017). Comparing a forward pass from layer l to layer $l + 1$ with activations a^l and connection weights W_{ij}^l in ANNs

$$a_i^{l+1} = \text{ReLU}\left(\sum_j W_{ij}^l a_j^l\right). \quad (2)$$

to the rate approximations of these activations a_l in SNNs as described in Section 2.1, we follow Rueckauer et al. (2017) and define the number of operations for ANNs and SNNs as follows.

For ANNs, operations are defined as the sum of all multiply-add computations, and for SNNs, operations are defined as synaptic operations, i.e., the sum of all spikes processed by all neurons. The number of ANN operations is constant across samples and rollout frames and only depends on the size of the input frame. In contrast, for SNNs, the number of operations depends on how many spikes are generated during the execution of the network. The overall number of spikes typically grows with the number of simulation steps n_{sf} and the magnitude of ANN activations, while it decreases with the sparsity of activations. Thus, a smaller number of simulation steps n_{sf} in SNNs leads to better energy-efficiency, but also to a less accurate approximation of ANN activations, potentially reducing accuracy. Generally, real-valued multiply-add operations in ANNs are computationally more expensive than synaptic operations in SNNs, but on the other hand memory accesses are more structured for ANNs. This trade-off varies between different accelerators and neuromorphic chips. As an estimate, the energy per multiply-add operation for a recent FPGA architecture (Manolopoulos et al., 2016) is about 555–1295.4 pJ, while for neuromorphic devices, a synaptic operation consumes only 2.8–360 pJ (Thakur et al., 2018). Observe, that

our definition of simulation steps per frame n_{sf} is related to the number of simulation steps T_{tot} in Rueckauer et al. (2017) and Deng et al. (2020) by the number of rollout frames K as $T_{tot} = n_{sf} \cdot K$, i.e., either one can be used to quantify the tradeoff between energy-efficiency and accuracy.

3. RESULTS

In this section we demonstrate fast, accurate, and energy-efficient inference with SNNs on five different sequence processing tasks. First, a toy dataset is used to illustrate the concept of temporal integration via streaming rollouts in ANNs, and shows the energy-efficiency of converted SNNs (Section 3.1). Second, we apply our approach to event data recorded by an event-based camera in real-world driving scenes (Section 3.2) and showcase the trade-off between the latency of network responses, the classification accuracy, and energy-efficiency. Finally, we demonstrate state-of-the-art performance on the established N-MNIST, CIFAR10-DVS, and DvsGesture benchmarks for classification on event-streams (Sections 3.3 to 3.5).

3.1. Moving Rectangles

This synthetic dataset consists of sequences composed of two frames containing one rectangle each, and the task is to determine whether the rectangle has moved to the left or right in the second image. See **Figure 2A** for an example of both classes and **Appendix A.1** for more details. This frame-based toy dataset is a minimal example of temporal integration, because the direction of movement can only be inferred from the complete sequence, but not from a single image. For this example, we use the network graph as described in Section 2.3 with $N_l = 1$ that results in streaming rollouts with a spatio-temporal receptive field of duration $\tau = 3$. To demonstrate the effect of temporal integration, we train this rollout with $\tau = 6$ input frames (for details, see Section 2.3), of which the first and second half (three input frames each) comprises the first and second image of the pair of moving rectangles, respectively.

During ANN inference, the predictions of the first three outputs of this rollout are on chance level (see data points in the area with light gray shading in **Figure 2B**), because only the first $\tau = 3$ input frames comprising the first rectangle are seen by these outputs. The network outputs at rollout frames $k = 8$ to 10 (dark gray shading) retrieve information from both frames of the input pair and, hence, can integrate this information to perfectly classify the movement direction of the rectangle. Note that for the chosen network graph and streaming rollout, the first response of the network occurs at $k = 5$, which reflects the length and the temporal delay $l_s = 4$ of the shortest path between input and output of the rollout.

Figure 2B shows that the SNN's accuracy follows that of the ANN and results in comparable overall performance. However, the SNN accuracy is lagging behind the ANN accuracy in **Figures 2B,C**, by at least one rollout frame. Multiple reasons could cause this lag: First, the SNN accuracy is calculated by averaging predictions over the last n_{sf} simulation steps. Second, accuracies could be further delayed by information that is stored in the values of the membrane potentials and is carried from one

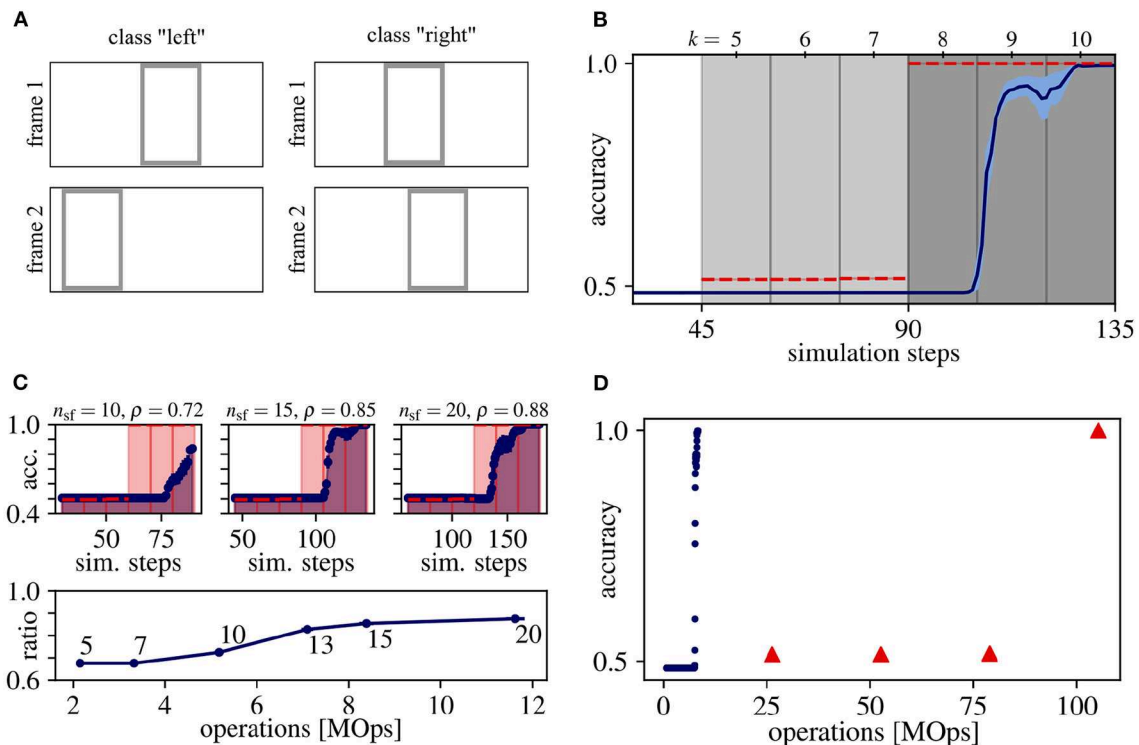


FIGURE 2 | Results for the moving rectangles dataset. **(A)** Samples for each class. **(B)** Average classification accuracy over rollout frames and corresponding simulation steps for ANN (red dashed line) and SNN (blue solid line) with the standard error of the mean in lightly colored areas. Vertical gray lines separate rollout frames and the shading of the background indicates, which image of the input sequence F_{in} is seen by the network output y : Light gray for the first and dark gray for both input frames. Note that the first prediction occurs at the fourth rollout frame after input onset (simulation step = 45), since the shortest path from input to output has length $l_s = 4$. **(C)** Determining the trade-off between accuracy and energy-efficiency. (Top) Same as **(B)**, but measured on 1,280 random samples of the training set and for three different values of n_{sf} . (Bottom) The ratio ρ between the area under curve of the ANN (shaded red) and SNN (shaded blue) for different values of n_{sf} over the number of operations. The number next to each datum is its respective number of simulation steps per frame n_{sf} . The accuracy ratio ρ saturates at $n_{sf} = 15$, which we therefore consider as a good trade-off between accuracy and energy-efficiency. **(D)** Average classification accuracy over the number of operations for ANN (red) and SNN (blue). For all data points in **(B–D)**, averages over 10 trials are plotted. Standard error of the means are always plotted, but sometimes too small to be visible.

to the next rollout frame, which can slow down the convergence of the approximation of firing rates.

To be able to execute the SNN, we have to determine the number of simulation steps per frame n_{sf} . Observe, that this hyperparameter comes with a trade-off between accuracy and energy-efficiency: With increasing n_{sf} , the rate approximation error decreases, leading to higher accuracies but also to a higher number of operations, decreasing energy-efficiency. In order to determine a good trade-off between accuracy and energy-efficiency, we sweep over different values for n_{sf} using 1,280 randomly chosen samples from the training set. The area under the curve is calculated for both the ANN and SNN accuracy over simulation steps, and the *accuracy ratio* ρ is calculated as the ratio between these areas. The difference between the accuracies of ANNs and SNNs decreases, i.e., ρ increases, with larger n_{sf} (see **Figure 2C**). Note that reaching $\rho = 1$ implies that ANN activations would have to be approximated by SNN firing rates instantaneously, i.e., within one simulation step, which is very unlikely in practice. The accuracy ratio ρ starts to saturate at $n_{sf} = 15$ simulation steps per rollout frame and, consequently,

we consider this value as a good trade-off for our experiments in **Figure 2D**.

In terms of efficiency, achieving the peak accuracy of 100% for this task requires 8.4 ± 0.6 MOps operations in the SNNs, which is approximately a factor of 13 lower compared to their ANN counterparts (105 MOps, see **Figure 2D**).

3.2. N-CARS

In this section, we apply our methods to real-world event-based vision data from driving scenes, for which the task is to classify the presence of a car in the recorded scene (Sironi et al., 2018). N-CARS uses event streams with continuous spike times for ON and OFF events that are triggered by positive and negative changes in light intensity, respectively. For this experiment, we choose ANN rollouts with $N_l = 5$ layers per block resulting in $\tau = 16$ input slots for the rollout and a spatio-temporal receptive field of duration $\tau = 16$ (for details, see **Appendix A**).

Our goal is to obtain good early predictions without sacrificing accuracies at later outputs (as already discussed in Section 2.2). This enables us to use early outputs for fast but relatively

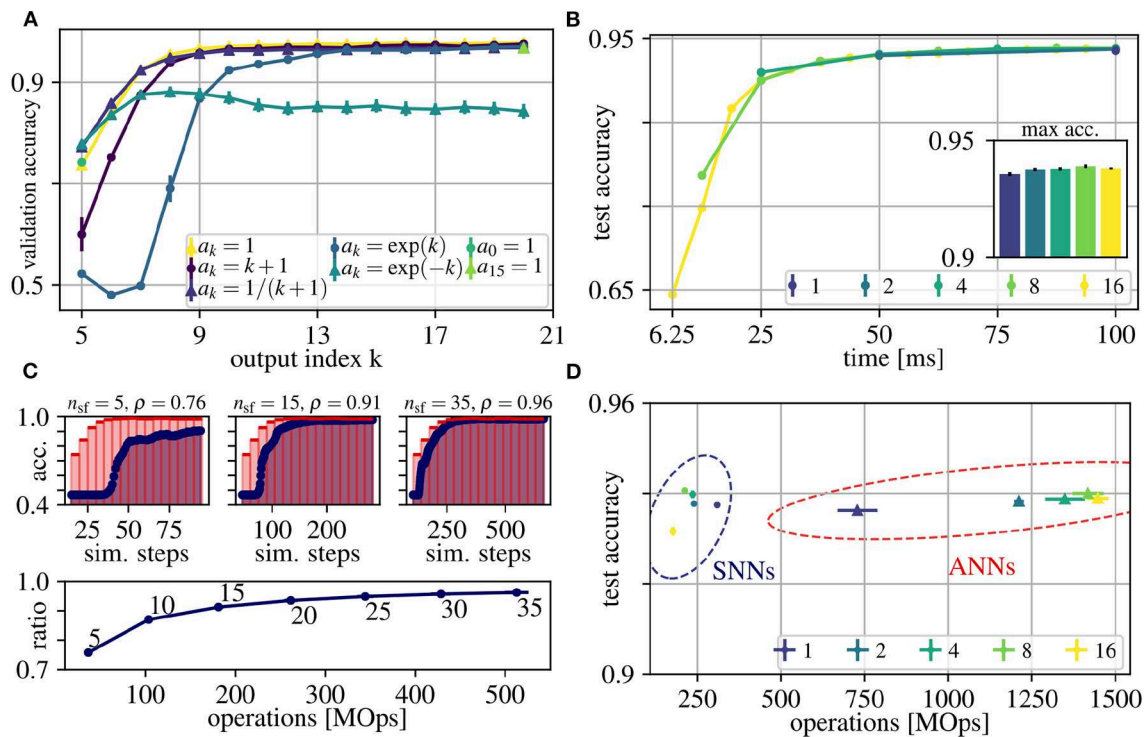


FIGURE 3 | Results for the N-CARS dataset. **(A)** Average validation accuracies over the index k of the outputs of the network rollouts. Different choices for weighting these outputs in the loss function are depicted with different colors (for details, see legend and Section 3). **(B)** Average accuracies of ANNs over real time of the input sequences for different values for N (see legend). The inset shows the maximum accuracies. **(C)** Determining the trade-off between accuracy and energy-efficiency. (Top) Accuracy of ANN and converted SNN over simulation time for $N = 16$ and $n_{sf} \in \{5, 15, 35\}$. Note that the first prediction occurs at the fourth rollout frame after input onset, since the shortest path from input to output has the length $l_s = 4$. (Bottom) The ratio ρ between the area under curve of the ANN (shaded red) and SNN (shaded blue) for different values of n_{sf} over the number of operations. The number next to each datum is its respective number of simulation steps per frame n_{sf} . The accuracy ratio ρ saturates at $n_{sf} = 15$, which we therefore consider as a good trade-off between accuracy and energy-efficiency. **(D)** Average peak accuracies over number of operations for the ANNs of **(B)** and the converted SNNs [same color coding as in **(B)**]. For all shown data, the error of the mean values are plotted after averaging over 10 trials, but are often too small to be seen.

inaccurate results and later outputs for slower results with higher accuracy. This trade-off between early and late performance can be tuned by the factors a_k of the loss function, which weight the losses from outputs at rollout frames $k = 5$ to $k = 21$ (see Eq. (1)). The index starts at $k = 5$, because the shortest path through the network is $l_s = 5$. In order to determine a good choice for a_k that achieves good early and late performance, we evaluated seven different options. **Figure 3A** shows resulting accuracies for each output k separately, after training, for the following proposals of sets of a_k :

- $a_k = 1$: Uniform weighting: the factor is identical for all k , such that early and late accuracy is considered equally important
- $a_k = k + 1$: Linearly increasing weighting: emphasizing late performance
- $a_k = \exp(k)$: Exponentially increasing weighting: even stronger emphasis on late performance
- $a_0 = 1$, others 0: Only consider first output
- $a_k = 1/(k+1)$: Moderately decreasing weighting: Emphasizing early performance
- $a_k = \exp(-k)$: Exponentially decreasing weighting: even stronger emphasis on early performance

- $a_{15} = 1$, others 0: Only consider last output

In practice, we normalize each set of a_k such that $\sum_k a_k = 1$, to avoid influencing the learning rate. Note that we share weights over time and, consequently, the very same weights have to fulfill multiple objectives at once, which could potentially deteriorate the accuracy of the network outputs. Weighting early outputs higher improves the early accuracy by up to 4.3% compared to constant a_k ($77.9 \pm 0.2\%$ for $\exp(-k)$ vs. $73.6 \pm 0.8\%$ for $a_k = 1$). However, increasing the accuracy for early outputs degrades late performance by up to 13.5% ($\exp(-k)$). This effect can be explained by the trade-off between using the available capacity of the network to decrease the loss at early outputs and to provide meaningful features for further processing required to decrease the loss at later outputs. Weighting early outputs much higher than late outputs ($\exp(-k)$) shifts this trade-off toward using most of the network capacity for instant classification and suppressing feature generation for later outputs and temporal integration. In all cases in which the late performance was prioritized (including $a_k = 1$), the maximum accuracies are similar to each other ($< 0.04\%$ difference). In conclusion, a uniform weighting $a_k = 1$ represents a good trade-off

between early and late performance indicating synergies between generating rich features required to achieve peak performance and generating sufficient features for immediate classification. Other choices for the weighting a_k either result in lower peak or lower early accuracy (see **Figure 3A**).

For ANN training, each event-stream of duration 100 ms is divided into N input frames and in the following we investigate the impact of different choices for N on accuracy, latency and energy-efficiency. The value of each pixel of these input frames is computed by averaging the firing rate during the resulting time intervals T_F (see also Section 2.3). The sampling frequency $1/T_F$ of event streams has to be chosen high enough to resolve the temporal information present in the input sequence and to achieve higher spatial sparsity of the generated input frames, but higher frequencies also cause lower signal-to-noise ratios due to fewer events per input frame. In real-time scenarios, higher sampling frequencies, i.e., smaller T_F , result in more frequent and earlier network responses (see **Figure 3B**). For example for $N = 16$, the first prediction occurs at $T_F = 100 \text{ ms}/16 = 6.25 \text{ ms}$, while for $N = 1$ the first prediction occurs at $T_F = 100 \text{ ms}$. Note that for simplicity, we assume the computation of the network output y to be instantaneous in **Figure 3B**, and hence, the shown time axis reflects the time scale of the input sequence. Using the classification accuracies as evaluation criteria, we find $N = 16$ to reach the same accuracy with a higher temporal resolution as other frame intervals (see **Figure 3B**). For all N , we observe that the first network outputs (leftmost data points in **Figure 3B**) are already above chance level, although they only see the first input frame via the shortest path of the network. These early approximate predictions are then refined in later rollout frames, in which deeper networks can integrate information over multiple input frames. Overall, although sampled with different frequencies, the peak accuracy is almost the same across different N . This indicates that, for the N-CARS dataset, the information encoded in time is much less important than the spatial information. We conclude that the information present in the first 25 ms is already sufficient for a successful classification close to peak accuracy. This can be observed in the example given in **Figure 1A**, where the shape of the car can be distinctively identified after 25 ms.

After conversion, we have to choose the simulation steps per rollout frame n_{sf} , which significantly influences the accuracy and energy-efficiency of our network. As in Rueckauer et al. (2017), the approximation error of activations in ANNs by firing rates in SNNs increases over time. However, in our case this only holds on a per-frame basis, i.e., in our case the approximation error depends on the number of simulation steps per frame n_{sf} . Like for the moving rectangles dataset, we measure the accuracy ratio ρ over n_{sf} to find a good trade-off between energy-efficiency and accuracy. The accuracy ratio ρ starts to saturate at $n_{sf} = 15$ simulation steps per rollout frame (see **Figure 3C**) and, consequently, we consider this value as a good trade-off for our experiments in **Figure 3D**. For smaller n_{sf} , mostly the early accuracy (e.g., between simulation step 50 and 150 in **Figure 3C**) suffers, which can be explained as follows: First, SNNs perform worse than ANNs, because for early network

TABLE 1 | Average accuracies for the N-CARS dataset for ANNs and SNNs (10 trials each).

N-CARS	Acc.	# params	# ops [MOps]
HATS/linear SVM (Sironi et al., 2018)	90.2	–	–
Rec. U-Net+CNN (Rebecq et al., 2019)	91.0	$> 10^6$	–
ResNet-34 (Gehrig et al., 2019)	92.5	10^7	–
Streaming rollout ANN (ours)	94.00(± 0.05)	10^5	1,420(± 47)
Converted SNN (ours)	94.07 (± 0.05)	10^5	212.9(± 2.5)

Numbers in parentheses are standard errors of the mean values. # params are the number of parameters, i.e., the number of weights and biases of the network. # ops are the number of operations as defined in section 2.5. A minus sign indicates that the number of parameters or operations could not be estimated from the information available in the respective reference. The highest accuracy is highlighted in bold.

outputs, spikes are present only in the short paths from input to output of the networks. Consequently, the overall spiking activity is low, slowing down the convergence of the firing rate approximations. Second, neurons are initialized with lowest (remember that $V \in [0, 1]$) membrane voltage $V(0) = 0$, and, hence, it takes a few simulation steps until the neuron can spike. Third, transmitting information from one layer to the next requires at least one simulation step, resulting in a linear increase of the delay from input to output with the number of layers. In our case, the shortest path from input to output has to pass four layers and, hence the information is delayed by four simulation steps, such that the ideal case of $r = 1$ is impossible to reach. Converting ANNs to SNNs and using our choice for n_{sf} during the simulations of SNNs, results in accuracies of SNNs that are comparable to their corresponding ANNs, but requiring less energy. Furthermore, the energy-efficiency increases with the number of input frames N , up to an 8-fold factor for $N = 16$. Overall, to the best of our knowledge, both our ANNs and SNNs achieve the currently best results on the N-CARS dataset (see **Table 1**). In addition, our network has 126 378 parameters for $N_1 = 5$, which is significantly lower than the other approaches.

Since objects, e.g., cars, in the N-CARS dataset only slightly move during the short duration of the recordings, the frames of the input sequence are similar to each other (e.g., see data sample in **Figure 1A**). For almost static input, intermediate activations do not vary between rollout frames, and, hence, activations of ANNs can be approximated over multiple rollout frames in SNNs. The low number $n_{sf} = 15$ of simulation steps per rollout frame supports this hypothesis. This might also explain, why the SNN peak performance is above the ANN performance (see **Table 1**), since SNNs intrinsically average activations over rollout frames and may thereby increase the signal-to-noise ratio of the network outputs.

3.3. N-MNIST

N-MNIST (Orchard et al., 2015) is a widely used benchmark dataset for SNNs, which allows a comparison of our approach to various alternatives. Each sample in the N-MNIST dataset consists of a digit from the MNIST dataset projected onto a white wall and recorded with an event-based vision sensor, while performing three quick movements (saccades). The challenge is

TABLE 2 | Average accuracies for the N-MNIST dataset for ANNs and SNNs (10 trials each).

N-MNIST	Acc.	# params	# ops [MOps]
SNN with backprop (Lee et al., 2016)	98.66	$2 \cdot 10^6$	–
SNN with backprop (Wu et al., 2019)	99.53	$2 \cdot 10^6$	–
HATS/linear SVM (Sironi et al., 2018)	99.1	–	–
Rec. U-Net+CNN (Rebecq et al., 2019)	98.3	$> 10^6$	–
Streaming rollout ANN (ours)	99.56 (± 0.01)	$3 \cdot 10^5$	3,500(± 360)
Converted SNN (ours)	99.54(± 0.01)	$3 \cdot 10^5$	460(± 38)

Numbers in parentheses are standard errors of the mean values. Columns like in **Table 1**. The highest accuracy is highlighted in bold.

that as the digit moves the active pixels overlap, which means that averaging events over longer time periods results in blurred images, and classification becomes more difficult. We use the same network as for N-CARS, but with $N = 32$ input frames and a growth factor $g = 15$, which was determined by sweeping over $g \in \{9, 12, 15, 18\}$. Competitive results compared to state-of-the-art methods are achieved (see **Table 2**) and the SNN after conversion is ~ 7 times more energy-efficient than its ANN counterpart.

As the number of parameters is not directly listed in the work we compare to, we estimate them from their experiment description: Rebecq et al. (2019) use a U-Net + ResNet18, which typically has 10^6 to 10^7 parameters. Gehrig et al. (2019) use a ResNet-34, which has $\sim 10^7$ parameters. In Lee et al. (2016), three layers with (2312, 800, 10) neurons are used, resulting in 1,857,600 parameters. In Wu et al. (2019), they list different network sizes but we expect their best result to be from their largest network listed in **Table 1** in their paper. Each convolutional layer has $C_{in}C_{out}k_xk_y$ parameters, with $C_{in/out}$ the number of input/output channels and k_i the kernel sizes. In total, we count 2,840,704 parameters. Our approach has 319,890 parameters for $N_l = 5$, which is significantly lower than the other approaches. Sharing weights over time and taking temporal integration into account through our rollout mechanisms allows reaching state-of-the-art accuracy with a small memory footprint.

3.4. Cifar10-DVS

The CIFAR10-DVS dataset (Li et al., 2017) consists of 10,000 images extracted from the popular CIFAR-10 dataset. Each of the 10 classes is represented by 1,000 images. Each of these images is scaled up and moves on a diamond-shaped trajectory on a screen. The scene is recorded by a DVS128 sensor for 1.298 ± 0.040 s (mean and standard deviation over all samples) corresponding to 6 repetitions of the trajectory. The monitor's refresh rate of 60 Hz is filtered out of the event stream after recording. The dataset is split randomly into a training (90%) and test (10%) set while maintaining the balance of classes in each set. Then, the training set is further randomly split into a validation (20%) and new training (80%) set. After each training epoch, the accuracy on the validation set is calculated to determine the best model to be used for testing.

TABLE 3 | Average accuracies for the CIFAR10-DVS dataset for ANNs and SNNs (10 trials each).

CIFAR10-DVS	Acc.	# params	# ops [MOps]
HATS/linear SVM (Sironi et al., 2018)	52.4	–	–
SNN with backprop (Wu et al., 2019)	60.5	$2 \cdot 10^6$	–
Streaming rollout ANN (ours)	66.75 (± 0.22)	$5 \cdot 10^5$	8,800($\pm 1,300$)
Converted SNN (ours)	65.61(± 0.20)	$5 \cdot 10^5$	1,551(± 65)

Numbers in parentheses are standard errors of the mean values. Columns like in **Table 1**. The highest accuracy is highlighted in bold.

The data is pre-processed by cutting out the first 1.3 s of the event stream and splitting each sample into 48 frames resulting in $T_F = 1.3/48 = 27.08$ ms. Each edge of the diamond shape is, therefore, resolved by $48/6/4 = 2$ frames. To enable faster training and inference, the spatial resolution of each frame is reduced from 128×128 pixels to 32×32 pixels by bilinear interpolation. We use the same hyperparameters for the network architecture and training as for N-CARS and N-MNIST, but optimize the growth factor by training networks with $g \in \{9, 12, 18, 22\}$ and evaluating their accuracy on the validation set. Networks with $g = 18$ result in the best mean accuracy on the validation set, resulting in 480,852 parameters. The ANN-to-SNN conversion is done like for N-CARS and N-MNIST and $n_{sf} = 60$ simulation steps are found to be a good trade-off between accuracy and energy-efficiency. The accuracy of our approach is better than of any other approach for ANNs and SNNs reported to date, and our SNNs require 5-fold less operations than their corresponding ANNs (see **Table 3**).

3.5. DvsGesture

DvsGesture (Amir et al., 2017) is an action recognition dataset, where multiple participants performed 11 different gestures under varying lighting conditions. The gestures have an average duration of 6.5 ± 1.7 s and are recorded with a DVS128 sensor. The last class is an arbitrary gesture that each participant came up with. Because this class is not clearly defined, we train networks both with and without this additional target class, which has also been done in the approaches we compare to. We use the original dataset split of (Amir et al., 2017) and generate a validation set by randomly selecting 10% of the training set.

To simplify training and testing we follow the approach by Shrestha and Orchard (2018) and use only the first 1.5 s of each sample, which still contains multiple repetitions of the gesture. We split each sample into 240 frames corresponding to a frame interval of $T_F = 240/1.5 = 6.25$ ms. As for CIFAR10-DVS, we reduce the spatial dimension of each frame from 128×128 to 32×32 pixels. Inspired by Amir et al. (2017), we use stacks of 10 consecutive frames as input for each rollout frame in both ANNs and converted SNNs, such that each of the 24 inputs has 20 channels (as each frame has an on and off channel). This enables temporal integration over longer time scales without introducing motion blur in the individual frames. Note that the frequency of predictions is reduced to only every ten frames, i.e., every $10 \cdot 6.25$ ms = 62.5 ms. We use the same hyperparameters as

TABLE 4 | Average accuracies for the DvsGesture dataset for ANNs and SNNs (10 trials each).

DvsGesture	Acc.	# params	# ops [MOps]
10 CLASSES			
SNN on TrueNorth (Amir et al., 2017)	96.7	$1.5 \cdot 10^6$	–
SNN with backprop (Shrestha and Orchard, 2018)	93.64(±0.49)	–	–
PointNet-like ANN (Wang et al., 2019)	97.08	–	–
Streaming rollout ANN (ours)	97.16(±0.11)	$5 \cdot 10^5$	8,150(±740)
Converted SNN (ours)	96.97(±0.17)	$5 \cdot 10^5$	651(±43)
11 CLASSES			
SNN on TrueNorth (Amir et al., 2017)	94.59	$1.5 \cdot 10^6$	–
PointNet-like ANN (Wang et al., 2019)	95.32	–	–
Streaming rollout ANN (ours)	95.68(±0.32)	$8 \cdot 10^5$	15,000(±1,000)
Converted SNN (ours)	95.56(±0.14)	$8 \cdot 10^5$	931(±24)

Numbers in parentheses are standard errors of the mean values. Columns like in **Table 1**. The highest accuracy is highlighted in bold.

for CIFAR10–DVS, except for the growth factor. We sweep over $g \in \{6, 9, 12, 15, 18\}$ and find $g = 9$ to perform best for 10 classes and $g = 12$ for 11 classes. Our networks therefore have 476,460 and 821,820 parameters, respectively. ANN and SNN accuracies are on par with other state-of-the-art approaches (**Table 4**). For our SNNs, the number of operations is ~ 12.5 times lower than for the corresponding ANNs.

We calculate the number of parameters of Amir et al. (2017) from their **Table 1** as $\text{params} = \sum_{i=1}^{16} \text{feat}[i] \cdot \text{kernel}_x[i] \cdot \text{kernel}_y[i] \cdot \text{feat}[i-1]/\text{groups}[i] = 1,528,536$. Shrestha and Orchard (2018) do not provide a detailed network description for their DvsGesture experiments.

4. DISCUSSION

We have presented a novel way of training efficient SNNs for sequence processing via conversion from ANNs. The crucial observation is the connection between axonal delays in the SNN and the rollout strategy in the ANN. Streaming rollouts of ANNs are shown to be a particularly good fit, as they closely resemble the fully model-parallel execution in SNNs. To unify the two approaches, we introduced several additions to the existing conversion approach, such as a more general weight rescaling scheme, a new way to calculate predictions in the SNN, rescaling of average pooling layers and axonal delays. As a result, we make ANN-to-SNN conversion applicable in a principled manner to input signals changing over time, including general time series and the special case of event-based input data. Due to the fact that

the streaming rollout imposes constraints on the ANN during training our approach can be interpreted as a “constrain-then-train” approach for SNNs (Esser et al., 2015; Pfeiffer and Pfeil, 2018), for which the superior training mechanisms available for ANNs are combined with the efficiency of SNN execution.

We identify and highlight in our experiments particular advantages of applying conversion to rolled-out networks. Our proposed training and conversion scheme results in SNNs that efficiently integrate temporal information, provide early approximate network outputs, and achieve state-of-the-art results on the N-MNIST, N-CARS, DvsGesture and CIFAR10–DVS datasets with smaller networks than other approaches, and with SNNs that are consistently more energy-efficient than their ANN counterparts. A uniform weighting of the network outputs in the loss function enables good early and late performance compared to other weighting patterns, such that even for the first network output, the prediction is significantly above chance level. Our framework is flexible enough to allow different trade-offs between early and late performance by choosing different weight factors a_k . In this study, for the first time, streaming rollouts were applied to realistic and large-scale time series data, and were shown to be competitive with other approaches on multiple widely used event-based vision tasks (see **Tables 1–4**).

Although we use only delays of one rollout frame in our experiments, in principle, arbitrary delays can be incorporated into the network rollouts. This principle is useful to convert advanced ANN architectures with temporal convolutions (van den Oord et al., 2016b; Bai et al., 2018) that require multiple delays when rolled out. This is a big advantage over previous conversion approaches (e.g., Cao et al., 2015; Rueckauer et al., 2017), which do not take delays of connections into account. For purely feed-forward SNNs on suitable hardware (Farabet et al., 2012; Pérez-Carrasco et al., 2013) a pseudo-simultaneous spread of information, i.e., all delays in the network are zero, is advantageous, but causes de-synchronization if information needs to be integrated over time. Our approach generalizes the work of Diehl et al. (2016), who have shown a conversion approach for Elman-type recurrent networks using fixed delays in the recurrent layer and zero delays for feed-forward connections. Note that although our experiments only show DenseNet architectures and therefore lead to a linear growth of the size of the temporal receptive field with network depth, this is not a general restriction of our approach. More complex network graphs, for example containing temporal convolutions or recurrent connections, lead to a super-linear growth of the temporal receptive field.

Rescaling weights and biases during conversion by using percentiles instead of maximum values as upper limits for ANN activations increases the accuracy. However, the percentile values of activations calculated over all rollout frames may overestimate the size of ANN activations in single rollout frames and would, hence, decrease the effective resolution of the firing rate approximation (for details, see Section 2.4). For example, in our network rollouts, the activity increases with each rollout frame (which does not hold in general). This results in strong overestimations of activations for early network outputs, which in turn increase approximation errors and, hence, decrease

accuracy (see e.g., **Figure 3C**). As the activity increases with more and more paths from input to output of the network contributing to later network outputs, this approximation error decreases until an optimal effective resolution is reached when spiking activity is present in all parts of the network. Adaptively rescaling the SNN weights or firing thresholds could be a solution to alleviate this effect. This can be seen as a kind of homeostasis mechanism that keeps the overall firing rates of SNNs at a constant level.

Instead of simply averaging event rates to obtain input frames, our approach generalizes to using more advanced features for event-based vision, such as time surfaces (Sironi et al., 2018), event spike tensors (Gehrig et al., 2019) or motion-based features (Clady et al., 2017). As use-cases for event-based vision are becoming increasingly challenging (Gallego et al., 2019), and neuromorphic hardware platforms become more mature (DeBole et al., 2019), our approach fills an important gap to provide powerful SNNs ready for deployment on those platforms.

A major goal of our approach is achieving energy-efficiency, which we measure by the number of operations necessary to reach the desired performance. High efficiency during early inference is enabled by temporal skip connections and carefully choosing the weight factors a_k in the loss function to achieve a good early accuracy without deteriorating the later peak accuracy. After ANN-to-SNN conversion, the SNNs are consistently more energy-efficient than their corresponding ANNs, and the achieved relative gain in efficiency is higher than, e.g., reported by Rueckauer et al. (2017). This may be due to the different neural architectures and the increased sparsity of the input in our study. The sparsity of a single frame increases with a decreasing time interval T_F over which events are accumulated. To further increase the efficiency we ran multiple experiments including quantization and observed interesting dependencies between quantization levels, network architectures, energy-efficiency, and final accuracy. A thorough investigation would exceed the scope of this study and is left for future studies.

In summary, our approach sets a new standard for spiking neural networks for processing spatio-temporal event streams both in terms of accuracy and efficiency. However, in this study, information is encoded with firing rates, the underlying principle

of network conversions, and we did not exploit the potential of encoding information with spike times that potentially allow for even more energy-efficient solutions (for an overview, see Pfeiffer and Pfeil, 2018). We are excited to see our results as a competitive baseline for further studies in the direction of spike codes.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

AK and TP designed the study, contributed to the source code, conducted the experiments, and evaluated the results. MP and EC provided the feedback and scientific advice throughout the process. All authors contributed to the final manuscript.

FUNDING

This publication has received funding from the European Union's Horizon 2020 research innovation programme under grant agreement 732642 (ULPEC project). This research was supported by the Cluster of Excellence Cognitive Interaction Technology CITEC (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG).

ACKNOWLEDGMENTS

We acknowledge the financial support of the German Research Foundation (DFG) and the Open Access Publication Fund of Bielefeld University for the article processing charge.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fnins.2020.00439/full#supplementary-material>

REFERENCES

- Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 7243–7252. doi: 10.1109/CVPR.2017.781
- Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR* abs/1803.01271.
- 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
- Clady, X., Maro, J.-M., Barré, S., and Benosman, R. B. (2017). A motion-based feature for event-based pattern recognition. *Front. Neurosci.* 10:594. doi: 10.3389/fnins.2016.00594
- Davies, M., Srinivasa, N., Lin, T., 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
- DeBole, M. V., Taba, B., Amir, A., Akopyan, F., Andreopoulos, A., Risk, W. P., et al. (2019). Truenorth: accelerating from zero to 64 million neurons in 10 years. *Computer* 52, 20–29. doi: 10.1109/MC.2019.2903009
- Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., et al. (2020). Rethinking the performance comparison between SNNs and ANNs. *Neural Netw.* 121, 294–307. doi: 10.1016/j.neunet.2019.09.005
- 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 *2016 IEEE International Conference on Rebooting Computing (ICRC)* (San Diego, CA), 1–8. doi: 10.1109/ICRC.2016.7738691
- Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems* (Montreal, QC), 1117–1125.
- Farabet, C., Paz, R., Pérez-Carrasco, J., Zamarreño, C., Linares-Barranco, A., LeCun, Y., et al. (2012). Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing. *Front. Neurosci.* 6:32. doi: 10.3389/fnins.2012.00032

- Fischer, V., Koehler, J., and Pfeil, T. (2018). "The streaming rollout of deep networks-towards fully model-parallel execution," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Montreal, QC: Curran Associates, Inc.), 4039–4050.
- Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2013). Overview of the spinnaker system architecture. *IEEE Trans. Comput.* 62, 2454–2467. doi: 10.1109/TC.2012.142
- Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: a survey. *CoRR abs/1904.08405*.
- Gehrig, D., Loquercio, A., Derpanis, K. G., and Scaramuzza, D. (2019). "End-to-end learning of representations for asynchronous event-based data," in *The IEEE International Conference on Computer Vision (ICCV)* (Seoul). doi: 10.1109/ICCV.2019.00573
- Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9781107447615
- Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. (2018). "Condensenet: an efficient densenet using learned group convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Salt Lake City, UT). doi: 10.1109/CVPR.2018.00291
- Huang, G., Liu, Z., and Weinberger, K. Q. (2017). "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Honolulu, HI). doi: 10.1109/CVPR.2017.243
- 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, 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
- Liu, S.-C., and Delbruck, T. (2010). Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* 20, 288–295. doi: 10.1016/j.conb.2010.03.007
- Manolopoulos, K., Reisis, D., and Chouliaras, V. (2016). An efficient multiple precision floating-point multiply-add fused unit. *Microelectron. J.* 49, 10–18. doi: 10.1016/j.mejo.2015.10.012
- Martí, D., Rigotti, M., Seok, M., and Fusi, S. (2015). Energy-efficient neuromorphic classifiers. *Neural Comput.* 28, 2011–2044. doi: 10.1162/NECO_a_00882
- 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
- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). *Surrogate Gradient Learning in Spiking Neural Networks*. New York, NY: IEEE.
- 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
- 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
- Osswald, M., Ieng, S.-H., Benosman, R., and Indiveri, G. (2017). A spiking neural network model of 3d perception for event-based neuromorphic stereo vision systems. *Sci. Rep.* 7:40703. doi: 10.1038/srep40703
- 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
- Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774
- Posch, C., Matolin, D., and Wohlgenannt, R. (2010). "High-DR frame-free PWM imaging with asynchronous AER intensity encoding and focal-plane temporal redundancy suppression," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2430–2433. doi: 10.1109/ISCAS.2010.5537150
- 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
- Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. (2019). Events-to-video: bringing modern computer vision to event cameras. *CoRR abs/1904.08298*. doi: 10.1109/CVPR.2019.00398
- Rieke, F., Warland, D., Ruyter, R. D., Steveninck, V., and Bialek, W. (1999). *Spikes: Exploring the Neural Code*. Cambridge, MA: MIT Press.
- 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
- Schemmel, J., Brüderle, D., Gröbl, 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 (Paris)*, 1947–1950. doi: 10.1109/ISCAS.2010.5536970
- Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Montreal, QC: Curran Associates, Inc.), 1412–1421.
- Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "HATS: Histograms of averaged time surfaces for robust event-based object classification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Salt Lake City, UT). doi: 10.1109/CVPR.2018.00186
- Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018). Large-scale neuromorphic spiking array processors: a quest to mimic the brain. *Front. Neurosci.* 12:891. doi: 10.3389/fnins.2018.00891
- Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). "Efficient object localization using convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA). doi: 10.1109/CVPR.2015.7298664
- van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016a). Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*.
- van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel recurrent neural networks. *CoRR abs/1601.06759*.
- Wang, Q., Zhang, Y., Yuan, J., and Lu, Y. (2019). "Space-time event clouds for gesture recognition: From RGB cameras to event cameras," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Honolulu, HI), 1826–1835. doi: 10.1109/WACV.2019.00199
- Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560. doi: 10.1109/5.58337
- 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. (2019). "Direct training of spiking neural networks: faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence* (Honolulu, HI). doi: 10.1609/aaai.v33i01.33011311
- Zhang, Z., Liang, X., Dong, X., Xie, Y., and Cao, G. (2018). A sparse-view ct reconstruction method based on combination of densenet and deconvolution. *IEEE Trans. Med. Imaging* 37, 1407–1417. doi: 10.1109/TMI.2018.2823338
- Zhu, Y., and Newsam, S. (2017). "Densenet for dense flow," in *2017 IEEE International Conference on Image Processing (ICIP)* (Beijing), 790–794. doi: 10.1109/ICIP.2017.8296389

Conflict of Interest: MP, TP, and AK were employed by Robert Bosch GmbH.

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

Copyright © 2020 Kugele, Pfeil, Pfeiffer and Chicca. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)

Jacques Kaiser¹, Hesham Mostafa² and Emre Neftci^{3,4*}

¹ FZI Research Center for Information Technology, Karlsruhe, Germany, ² Department of Bioengineering, University of California, San Diego, La Jolla, CA, United States, ³ Department of Cognitive Sciences, University of California, Irvine, Irvine, CA, United States, ⁴ Department of Computer Science, University of California, Irvine, Irvine, CA, United States

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

James Courtney Knight,
University of Sussex, United Kingdom
Yulia Sandamirskaya,
Intel, Germany

*Correspondence:

Emre Neftci
eneftci@uci.edu

Specialty section:

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

Received: 27 November 2019

Accepted: 07 April 2020

Published: 12 May 2020

Citation:

Kaiser J, Mostafa H and Neftci E
(2020) Synaptic Plasticity Dynamics
for Deep Continuous Local Learning
(DECOLLE). *Front. Neurosci.* 14:424.
doi: 10.3389/fnins.2020.00424

A growing body of work underlines striking similarities between biological neural networks and recurrent, binary neural networks. A relatively smaller body of work, however, addresses the similarities between learning dynamics employed in deep artificial neural networks and synaptic plasticity in spiking neural networks. The challenge preventing this is largely caused by the discrepancy between the dynamical properties of synaptic plasticity and the requirements for gradient backpropagation. Learning algorithms that approximate gradient backpropagation using local error functions can overcome this challenge. Here, we introduce Deep Continuous Local Learning (DECOLLE), a spiking neural network equipped with local error functions for online learning with no memory overhead for computing gradients. DECOLLE is capable of learning deep spatio temporal representations from spikes relying solely on local information, making it compatible with neurobiology and neuromorphic hardware. Synaptic plasticity rules are derived systematically from user-defined cost functions and neural dynamics by leveraging existing autodifferentiation methods of machine learning frameworks. We benchmark our approach on the event-based neuromorphic dataset N-MNIST and DvsGesture, on which DECOLLE performs comparably to the state-of-the-art. DECOLLE networks provide continuously learning machines that are relevant to biology and supportive of event-based, low-power computer vision architectures matching the accuracies of conventional computers on tasks where temporal precision and speed are essential.

Keywords: spiking neural network, embedded learning, neuromorphic hardware, surrogate gradient algorithm, backpropagation

1. INTRODUCTION

Understanding how the plasticity dynamics in multilayer biological neural networks are organized for efficient data-driven learning is a long-standing question in computational neurosciences (Sussillo and Abbott, 2009; Clopath et al., 2010; Zenke and Ganguli, 2017). The generally unmatched success of deep learning algorithms in a wide variety of data-driven tasks prompts the question of whether the ingredients of their success are compatible with their biological counterparts, namely Spiking Neural Networks (SNNs). Biological neural networks distinguish themselves from Artificial Neural Networks (ANNs) by their continuous-time dynamics, the locality of their operations (Baldi et al., 2017), and their spike(event)-based communication.

Taking these properties into account in a neural network is challenging, as the spiking nature of the neurons' nonlinearity makes it non-differentiable, the continuous-time dynamics raise a temporal credit assignment problem and the assumption of computations being local to the neuron disqualifies the use of Back-Propagation-Through-Time (BPTT).

In this article, we describe DECOLLE, a SNN model with plasticity dynamics that solves the three problems above, and that performs at proficiencies comparable to that of multilayer neural networks. DECOLLE uses layerwise local readouts (Mostafa et al., 2017), which enables gradients to be computed locally (Figure 1). To tackle the temporal dynamics of the neurons, we use a recently established equivalence between SNNs and recurrent ANNs (Neftci et al., 2019). This equivalence rests on a computational graph of the SNN, which can be implemented with standard machine learning frameworks as a recurrent neural network. Unlike BPTT and like Real-Time Recurrent Learning (RTRL) (Williams and Zipser, 1989), DECOLLE is formulated in a way that the information necessary to compute the gradient is propagated forward, making the plasticity rule temporally local. Existing rules of this sort require dedicated state variables for every synapse, thus scaling at least quadratically with the number of neurons (Williams and Zipser, 1989; Zenke and Ganguli, 2017). In contrast, DECOLLE scales linearly with the number of neurons. This is achieved using a spatially and temporally local cost function reminiscent of readout mechanisms used in liquid state machines (Maass et al., 2002), but where the readout is performed over a fixed random combination of the neuron outputs. Our approach can be viewed as a type of synthetic gradient, a technique used to decouple one or more layers from the rest of the network to prevent layerwise locking (Jaderberg et al., 2016). Although synthetic gradients usually involve an outer loop that is equivalent to a full Back-Propagation (BP) through the network, DECOLLE instead relies on the random initialization of the local readout and forgoes the outer loop.

Conveniently, DECOLLE can leverage existing autodifferentiation tools of modern machine learning frameworks. Its linear scalability enables the training of hundreds of thousands of spiking neurons on a single GPU, and continual learning on very fine time scales. We demonstrate our approach on the classification of gestures, the IBM DvsGesture dataset (Amir et al., 2017), recorded using an event-based neuromorphic sensor and report comparable performance to deep neural networks and even networks trained with BPTT.

1.1. Related Work

Previous work demonstrated learning in multiple layers of SNN using feedback alignment (Lillicrap et al., 2016; Neftci et al., 2017), performing at about 2% classification error on MNIST. However, those networks operated in the firing rate regime, by using either large populations or slow dynamics. In those works, training was not insensitive to the temporal dynamics of the neurons. The need for temporal dynamics are often obfuscated by the static nature of the benchmarked problems (e.g., MNIST), and a long readout interval that allows to ignore initial transients caused by the dynamics. In our previous work (Neftci et al.,

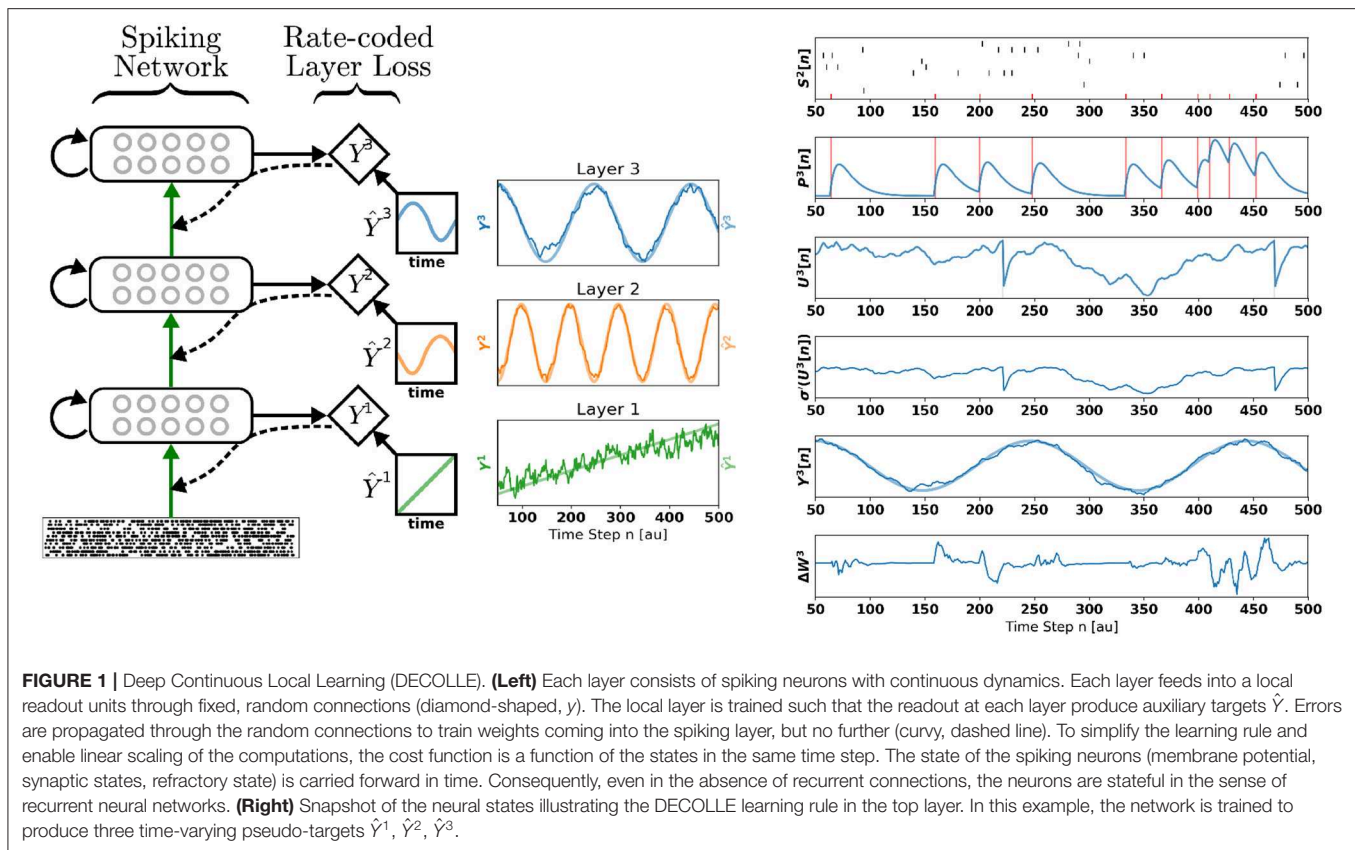
2017), ignoring temporal dynamics raised a "loop duration" problem, i.e., that the errors are available only after they have propagated through the network. This introduces latency or requires additional buffers for storing intermediate neural states. In traditional deep learning, the loop duration manifests itself as "layerwise locking," during which a layer's weights cannot be updated until a global cost function is evaluated (Jaderberg et al., 2016). This causes under utilization of the computing resources and a slowdown in learning. Besides the loop duration problem, multilayer networks trained with feedback alignment cannot reach the performances of gradient BP, especially with deeper networks ($\geq 30\%$ accuracy drop on ImageNet compared to backpropagation; Bartunov et al., 2018).

The complex dynamics of spiking neurons is an important feature that can be exploited for learning spatiotemporal patterns. In a single layer of neurons, this feature can be leveraged using gradient descent, since it is applicable to the subthreshold dynamics of leaky Integrate & Fire (I&F) neurons (Bohte et al., 2000; Gütiğ and Sompolinsky, 2006). Because the I&F neuron output is non-differentiable, however, the application of these approaches to multiple layers is not straightforward. To deal with this problem, SuperSpike uses a surrogate network with differentiable activation functions to compute an approximate gradient (Zenke and Ganguli, 2017). The authors show that this learning rule is equivalent to a forward-propagation of errors using synaptic traces, and is capable of learning in hidden layers of feedforward multilayer networks.

Because the traces need to be computed for every trainable parameter, Superspike scales temporally and spatially as $O(N^2)$, where N is the number of neurons. While the complex biochemical processes at the synapse could account for the quadratic scaling, it prevents an efficient implementation in available hardware. Like SuperSpike, DECOLLE uses surrogate gradients to perform weight updates, but as discussed later, the cost function is local in time and space, such that only one trace per input neuron is required. This enables the algorithm to scale linearly in space. Furthermore, in DECOLLE the computation of the gradients can reuse the variables computed for the forward dynamics, such that learning has no additional memory overhead.

DECOLLE has some resemblance with reservoir networks, which are neural networks with fixed internal connectivity and trainable readout functions (Jaeger, 2001; Maass et al., 2002; Eliasmith and Anderson, 2004; Sussillo and Abbott, 2009). The local readout in DECOLLE acts like a decoder layer in the flavor of the linear readouts in reservoir networks. In contrary to reservoir networks, DECOLLE learns the internal weights, but the readout weights are random and fixed. The training of the internal weights allows the network to learn representations that are easier to classify inputs for subsequent layers (Mostafa et al., 2017).

Spiking neural networks can be viewed as a subclass of binary, recurrent ANNs (Neftci et al., 2019). In the ANN sense, they are recurrent even when all the connections are feed-forward because the neurons maintain a state that is propagated forward at every time step. Binary neural networks, where both activations and/or weights are binary were studied in deep



learning as a way to decrease model complexity during inference (Courbariaux et al., 2016; Rastegari et al., 2016). BPTT for training SNNs was investigated in Bohte et al. (2000), Lee et al. (2016), Huh and Sejnowski (2017), Shrestha and Orchard (2018), and Bellec et al. (2018). BPTT-based approaches provide an unbiased estimation of the gradients but at a cost in memory, because the entire sequence and resulting activity states are stored to compute gradients. Although the truncation of the sequences (as in truncated BPTT) can mitigate this problem, it is not adequate when discretizing continuous-time networks, such as the SNN (Neftci et al., 2019) because the sequences can consist of hundreds of steps. This is because the time constants and simulation timestep in SNNs are such that the truncation window must be much larger. For SNN simulations with biological time constants, it is common to use simulation time steps $\Delta t \leq 1\text{ms}$. Smaller time steps capture non-linear dynamics more accurately and determine the temporal precision of all produced spike times. Assuming $\Delta t = 1\text{ms}$ (as used in this work), and if relevant interactions occur at one second, this implies that the truncation window must be at about 1,000 timesteps. This significantly increases the complexity of BPTT in SNNs. In practice, the size of SNN trainable by BPTT is severely limited by the available GPU memory (Shrestha and Orchard, 2018). As we explain later in this article, DECOLLE requires an order T less memory resources compared to BPTT, where T is the sequence length. Hence, DECOLLE networks are generally not memory-limited. Furthermore, DECOLLE can be

formulated as a local, three-factor synaptic plasticity rule, and is thus amenable to implementation in dedicated, event-based (neuromorphic) hardware (Davies et al., 2018), and compatible with neurobiology.

Decoupled Neural Interfaces (DNI) were proposed to mitigate layerwise locking in training deep neural networks (Jaderberg et al., 2016). In DNIs, this decoupling is achieved using a synthetic gradient, a neural network that estimates the gradients for a portion of the network. In an inner loop, the network parameters are trained using the synthetic gradients, and in an outer loop, the synthetic gradient network parameters are trained using a full BP step. The gradient computed using local errors in DECOLLE described below can be viewed as a type of synthetic gradient, which ignores the outer loop to avoid a full BP step. Although ignoring the outer loop limits DECOLLE's adaptation of the features using errors from other layers, we find that the network performs at or above state-of-the-art accuracy on N-MNIST and DVS Gesture benchmark tasks.

A related method called E-prop was developed in parallel to DECOLLE (Bellec et al., 2019). The resulting learning rule in E-prop is of the same form as Superspike and DECOLLE. E-prop uses adaptive spiking Long Short Term Memory (LSTM) neurons to maintain a longer term memory. This generalization allows to solve tasks with long-term dependencies (similar to LSTMs) but requires maintaining one trace per synapse. These memory requirements quickly exceed the capabilities of modern GPUs, especially when applied to convolutional neural networks. Even

in neuromorphic hardware, maintaining a synapse-specific trace can incur a prohibitive cost in area and power (Huayaney et al., 2016; Davies et al., 2018). In DECOLLE, we focus on networks which do not incur any memory overhead for training, allowing to tractably train large networks.

This work builds on a combination of how gradients are dynamically computed in SuperSpike and local errors. We show in the methods section that this combination considerably reduces the computational requirements compared to a computing a global loss.

2. METHODS

2.1. Neuron and Synapse Model

The neuron and synapse models used in this work follow leaky, current-based I&F dynamics with a relative refractory mechanism. The dynamics of the membrane potential U_i of a neuron i is governed by the following differential equations:

$$\begin{aligned} U_i(t) &= V_i(t) - \rho R_i(t) + b_i, \\ \tau_{mem} \frac{d}{dt} V_i(t) &= -V_i(t) + I_i(t), \\ \tau_{ref} \frac{d}{dt} R_i(t) &= -R_i(t) + S_i(t), \end{aligned} \quad (1)$$

with $S_i(t)$ the binary value (0 or 1) representing whether neuron i spiked at time t . The separation of the membrane potential into two variables U and V is done here for implementations reasons only. Biologically, the two states can be interpreted as a special case of a two-compartment model, consisting of one dendritic (V) and one somatic (U) compartment (Gerstner et al., 2014, Chapter 6.4). The absence of dynamics for U can be interpreted as the special case when somatic capacitance is much smaller than the distal capacitance. A spike is emitted when the membrane potential reaches a threshold $S_i(t) = \Theta(U_i(t))$, where $\Theta(x) = 0$ if $x < 0$, otherwise 1 is the unit step function. The constant b_i represents the intrinsic excitability of the neuron. The refractory mechanism is captured with the dynamics of R_i : the neuron inhibits itself after firing, by a constant weight ρ . In contrast to standard I&F refractory mechanisms, a strong enough input can still induce the neuron to fire immediately after a spike. The factors τ_{ref} and τ_{mem} are time constants of the membrane and refractory dynamics, respectively. I_i denotes the total synaptic current of neuron i , expressed as:

$$\tau_{syn} \frac{d}{dt} I_i(t) = -I_i(t) + \sum_{j \in \text{pre}} W_{ij} S_j(t), \quad (2)$$

where W_{ij} is the synaptic weights between pre-synaptic neuron j and post-synaptic neuron i . Because V_i and I_i are linear with respect to the weights W_{ij} , The dynamics of V_i can be

rewritten as:

$$\begin{aligned} V_i(t) &= \sum_{j \in \text{pre}} W_{ij} P_{ij}(t), \\ \tau_{mem} \frac{d}{dt} P_{ij}(t) &= -P_{ij}(t) + Q_{ij}(t), \\ \tau_{syn} \frac{d}{dt} Q_{ij}(t) &= -Q_{ij}(t) + S_j(t). \end{aligned} \quad (3)$$

The states P and Q describe the traces of the membrane and the current-based synapse, respectively. For each incoming spike, the trace Q undergoes a jump of height 1 and otherwise decays exponentially with a time constant τ_{syn} . Weighting the trace Q_{ij} with the synaptic weight W_{ij} results in the Post-Synaptic Potentials (PSPs) of neuron i caused by input neuron j .

All efferent synapses with identical time constants have identical dynamics. By linearity of P and Q , the state of the synapse can be described by a single synaptic variable per pre-synaptic neuron (Brette et al., 2007). In the equation above, this is evident by the fact that P_{ij} and Q_{ij} are only driven by S_j , and so the index i can be dropped. This results in as many P and Q variables as there are pre-synaptic neurons, independently of the number of synapses. This strategy is commonly used in synapse circuits in neuromorphic hardware to reduce circuit area (Bartolozzi and Indiveri, 2006), and in software simulations of spiking neurons to improve memory consumption and computation time (Brette et al., 2007).

Discrete Spike Response Model of the Neuron and Synapse Dynamics

Because a computer will be used to simulate the dynamics, the dynamics are simulated in discrete time. We denote the simulation time step with Δt . We also make the layerwise organization of the network apparent with the superscript l denoting the layer to which the neuron belongs. The dynamical equations in Equations (1) and (3) are expressed in discrete time as:

$$\begin{aligned} U_i^l[t] &= \sum_j W_{ij}^l P_j^l[t] - \rho R_i^l[t] + b_i^l, & P_j^l[t + \Delta t] &= \alpha P_j^l[t] + (1 - \alpha) Q_j^l[t], \\ & & Q_j^l[t + \Delta t] &= \beta Q_j^l[t] + (1 - \beta) S_j^{l-1}[t], \\ S_i^l[t] &= \Theta(U_i^l[t]), & R_i^l[t + \Delta t] &= \gamma R_i^l[t] + (1 - \gamma) S_i^l[t], \end{aligned} \quad (4)$$

where the constants $\alpha = \exp(-\frac{\Delta t}{\tau_{mem}})$, $\gamma = \exp(-\frac{\Delta t}{\tau_{ref}})$, and $\beta = \exp(-\frac{\Delta t}{\tau_{syn}})$ reflect the decay dynamics of the membrane potential U , the refractory state R and the synaptic state Q during a Δt timestep. Note that Equation (4) is equivalent to a discrete-time version of the Spike Response Model (SRM₀) with linear filters (Gerstner and Kistler, 2002).

2.2. Deep Learning With Local Losses

Loss functions are almost always defined using the network output at the top layer. Assuming a global cost function $\mathcal{L}(S^N)$ defined on the spikes S^N of the top layer and targets \hat{Y} , the gradients with respect to the weights in layer l are:

$$\frac{\partial \mathcal{L}(S^N)}{\partial W_{ij}^l} = \frac{\partial \mathcal{L}(S^N)}{\partial S_i^l} \frac{\partial S_i^l}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l}. \quad (5)$$

The factor $\frac{\partial \mathcal{L}(S^N)}{\partial S_i^l}$ captures the backpropagated errors, i.e., how changing the output of neuron i in layer l modifies the global loss. This problem is known as the credit assignment problem. It generally involves non-local terms, including the activity of other neurons, their errors, and their temporal history. Thus, using local information only, a neuron in a deep layer cannot infer how a change in its activity will affect the top-layer cost. An increasing body of work is showing that approximations to the backpropagated errors in SNNs can allow local learning, for example in feedback alignment (Lillicrap et al., 2014). However, maintaining the history of the dynamics efficiently remains a challenging and open problem. While it is possible to use BPTT methods to compute these errors, this comes at a significant cost in memory and computation (Williams and Zipser, 1989), and is not consistent with the constraint of local information.

We address this conundrum using deep local learning (Mostafa et al., 2017). We focus on a form of deep local learning that attaches random readouts to deep layers and defines auxiliary cost functions over the readout. These auxiliary cost functions provide a task-relevant source of error for neurons in deep layers. The random readout is obtained by multiplying the neural activations with a random and fixed matrix. Training deep layers using auxiliary local errors that minimize the cost locally still allows the network as a whole to reach a small top-layer cost. As explained in Mostafa et al. (2017), minimizing a local readout's classification loss puts pressure on deep layers to learn useful task-relevant features, which allow the random local classifiers to solve the task. Moreover, each layer builds on the features of the previous layer to learn features that are further disentangled with respect to the categories for its local random classifier compared to the previous layer. Thus, even though no error information propagates downwards through the layer stack, the layers indirectly learn useful hierarchical features that end up minimizing the cost at the top layer. Although the reasons for the effectiveness of local errors in deep network is intriguing and merits further work, it is orthogonal to the scope of this article. In this article, we focus on the fact that, provided local loss functions, surrogate learning in deep spiking neural networks becomes particularly efficient.

2.3. Deep Continuous Local Learning (DECOLLE)

As discussed above, in DECOLLE, we attach a random readout to each of the N layers of spiking neurons:

$$Y_i^l = \sum_j G_{ij}^l S_j^l,$$

where G_{ij}^l are fixed, random matrices (one for each layer l) and Θ is an activation function. The global loss function is then defined as the sum of the layerwise loss functions defined on the random readouts, i.e. $\mathcal{L} = \sum_{l=1}^N L^l(Y^l)$. To enforce locality, DECOLLE sets to zero all non-local gradients, i.e., $\frac{\partial L^l}{\partial W_{ij}^m} = 0$ if $m \neq l$. With

this assumption, the weight updates at each layer become:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial W_{ij}^l} = -\eta \frac{\partial L^l}{\partial S_i^l} \frac{\partial S_i^l}{\partial W_{ij}^l}, \quad (6)$$

where η is the learning rate. Assuming the loss function depends only on variables in same time step, the first gradient term on the right hand side, $\frac{\partial L^l}{\partial S_i^l}$, can be trivially computed using the chain rule of derivatives. Applying the chain of derivatives to the second gradient term yields:

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \frac{\partial \Theta(U_i^l)}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l}.$$

Due to the sparse, binary activation of spiking neurons, this expression vanishes everywhere except at 0, where it is infinite (Nefci et al., 2019). To solve this problem, parameter updates in DECOLLE are based on a differentiable but slightly different version of the task-performing network. This approach was previously described as surrogate gradient-based learning (Zenke and Ganguli, 2017; Nefci et al., 2019):

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \sigma'(U_i^l) \frac{\partial U_i^l}{\partial W_{ij}^l},$$

where $\sigma'(U_i^l)$ is the surrogate gradient of the non-differentiable step function $\Theta(U_i^l)$. The rightmost term is computed as:

$$\frac{\partial U_i^l}{\partial W_{ij}^l} = P_j^l - \rho \frac{\partial R_i^l}{\partial W_{ij}^l}.$$

The terms involving R_i^l are difficult to calculate because they depend on the spiking history of the neuron. As in Superspike, we ignore these dependencies and use regularization to favor low firing rates, a regime in which the R_i^l has a negligible effect on the membrane dynamics. Putting all three terms together, we obtain the DECOLLE rule governing the synaptic weight update:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial S_i^l} \sigma'(U_i^l) P_j^l. \quad (7)$$

In the special case of the Mean Square Error (MSE) loss for layer l , described as

$$L^l = \frac{1}{2} \sum_i (Y_i^l - \hat{Y}_i^l)^2,$$

the DECOLLE rule becomes

$$\begin{aligned} \Delta W_{ij}^l &= -\eta \text{error}_i^l \sigma'(U_i^l) P_j^l, \\ \text{error}_i^l &= \sum_k G_{ki}^l (Y_k^l - \hat{Y}_k^l), \end{aligned} \quad (8)$$

where \hat{Y}^l is the pseudo-target vector for layer l .

2.3.1. Memory Complexity of DECOLLE

The variables P and U required for learning are local and readily available from the forward dynamics. Because the errors are computed locally to each layer, DECOLLE does not need to store any additional intermediate variables, i.e., there is no space requirement for the parameter update computation. The same neural traces P and Q maintained during the forward pass are sufficient (see section 2.4). The computational cost of the weight update is the same as the Widrow-Hoff rule (one addition and two products per connection, see Equation 8). This makes DECOLLE significantly cheaper to implement compared to BPTT for training SNN, e.g., SLAYER (Shrestha and Orchard, 2018) which scales spatially as $O(NT)$, where T is the number of timesteps (see **Appendix** section 5.2 for details on scaling).

2.3.2. Sign-Concordant Feedback Alignment in the Local Layers

The gradients of the local losses L_i^l involve backpropagation through the local random projection Y^l . This is a non-local operation as it requires the symmetric transpose of the matrix G . This raises a weight transport problem, whereby the synaptic weight must be “transported” from one neuron to another. In a von Neumann computer, this is not a problem since memory is shared across processes. However, if memory is local, then a dedicated process must transmit this information. Feedback alignment in non-spiking networks was demonstrated to overcome this problem at a cost in accuracy (Mostafa et al., 2017). In our experiments, we use sign-concordant feedback weights to compute the gradients of the local losses: the backward weights have the same sign as the forward ones, but subject to fixed multiplicative Gaussian noise. The noise here reflects the fact that weights do not need to be exactly symmetric. This assumption is the most plausible scenario in mixed-signal neuromorphic devices, where connections can be programmed with the same sign bidirectionally, but the effective weights are subject to fabrication mismatch (Neftci et al., 2011). Since the weights in the local readouts are fixed, there is no weight transport problem during learning. Thus, the computation of the errors can be carried out using another random matrix H^l (Lillicrap et al., 2016) whose elements are equal to $H_{ij}^l = G_{ij}^{l,T} \omega_{ij}^l$ with a Gaussian distributed $\omega_{ij}^l \sim N(1, \frac{1}{2})$. To enforce sign-concordance, all values ω_{ij}^l below zero were set to zero.

2.3.3. Biological Plausibility of DECOLLE and Suitability for Neuromorphic Hardware

Equation (8) consists of three factors, one modulatory ($error_i$), one post-synaptic [$\sigma'(U_i^l)$], and one pre-synaptic (P_j^l). These types of rules are often termed three-factor rules, which have been shown to be consistent with biology (Pfister et al., 2006), while being compatible with a wide number of unsupervised, supervised, and reinforcement learning paradigms (Urbanczik and Senn, 2014). The terms P and Q represent neural and synaptic states that are readily available at the neuron. In our previous work and general experience, the shape of the surrogate

function σ does not play a major role in DECOLLE¹. The surrogate function σ can be a piecewise linear function (Neftci et al., 2017), such that σ' becomes a boxcar function. This corresponds to a learning update that is gated by the post-synaptic membrane potential, and is reminiscent of membrane voltage-based rules, where spike-driven plasticity is induced only when membrane voltage is inside an eligibility window (Brader et al., 2007; Chicca et al., 2013).

In the derivation of the DECOLLE rule, we used an instantaneous readout function Y^l in the sense that it did not depend on states of the previous time step. In biology, this readout would be carried out by spiking neurons. This introduces a temporal dependency. As in SuperSpike, this temporal dependency significantly increases the complexity of the learning, and is costly to implement in neuromorphic hardware. One solution is to compute the errors using spiking neurons with dynamics faster than those of the hidden neurons. In mixed signal hardware, this can be achieved through fast membrane and synaptic time constants. In digital hardware this could be achieved using a dedicated logic block.

2.3.4. Regularization and Implementation Details

From a technological point of view, SNNs are interesting when the spike rate is low as dedicated neuromorphic hardware can directly exploit this sparsity to reduce computations by the same factor (Merolla et al., 2014; Davies et al., 2018). To ensure reasonable firing rates and prevent sustained firing, we use two regularizers. One keeps U below to the firing threshold on average, and one activity regularizer enforces a minimum firing rate in each layer. The final loss function is:

$$\mathcal{L}_g = \sum_l L^l + \lambda_1 \langle [U_i^l + 0.01]^+ \rangle_i + \lambda_2 \langle [0.1 - U_i^l]^+ \rangle_i \quad (9)$$

where $\langle \cdot \rangle_i$ denotes averaging over index i , $[\cdot]^+$ is a linear rectification, and λ_1, λ_2 are hyperparameters. The minimum firing rate regularization is included to prevent the layers becoming completely silent during the training. Our experiments used a piecewise linear surrogate activation function, such that its derivative becomes the boxcar function $\sigma'(x) = 1$ if $x \in [-0.5, 0.5]$ and 0 otherwise.

In all our experiments, weight updates are made for each time step of the simulation. We use the AdaMax optimizer (Kingma and Ba, 2014) with parameters $\beta_1 = 0, \beta_2 = 95$ and learning rate 10^{-9} , and a smooth L1 loss. Biases were used for all layers and trained in all DECOLLE layers. The weights G^l used for the local readouts were initialized uniformly. PyTorch code and a tutorial are publicly available on Github². DECOLLE is simulated using mini-batches to leverage the GPU's parallelism.

¹Conversely, the particular surrogate function is reported to play an important role in BPTT (Bellec et al., 2018). This is likely due the product of the gradient approximations carried across multiple layers. This in turn can cause vanishing or exploding gradients.

²<https://github.com/nmi-lab/decollle-public>

2.4. Computational Graph and Implementation Using Automatic Differentiation

Perhaps one of the strongest advantages of DECOLLE is its out-of-the-box compatibility with Automatic Differentiation (AD) tools for implementing gradient BP. AD is a technology recently incorporated in machine learning frameworks to automatically compute gradients in a computational graph³. AD operates on the principle that all numerical computations are compositions of a finite set of elementary operations for which derivatives are known. By combining the derivatives of the operations through the chain rule of derivatives, the derivative of the overall composition can be computed in a single pass (Baydin et al., 2017).

In practice, machine learning frameworks augment each elementary computation with its corresponding derivative function. As the desired operation is constructed, the dependencies with other variables are recorded as a computational graph. To perform gradient BP, after a forward pass, a backward pass computes all the derivatives of the operations in the graph. The root node of the reverse graph is typically a scalar loss function, and the leaf nodes are generally inputs and parameters of the network. After the backward pass, the gradients of all leaf nodes are applied to the trained parameters or inputs according to the optimization routine (e.g., Adam or similar).

SNNs being a special case of recurrent neural networks, it is possible to apply AD to the full graph (Shrestha and Orchard, 2018). On the other hand, DECOLLE only requires backpropagating through a subgraph corresponding to one layer and within the same time step (Figure 2). This is because the information necessary for computing the gradients (P , Q , R , and U) is carried forward in time, and because local loss functions provide gradients for each layer.

AD in DECOLLE thus computes the gradients, locally, for each layer within each timestep. Because some operations in the subgraph can be non-differentiable (such as the spiking nonlinearity), we call this the “surrogate gradient backprop” (Figure 2). This integration allows leveraging the layers, operations, optimizers and cost functions provided by the software. All experiments under the Experiments section use AD to compute derivatives. To prevent AD from unnecessarily backpropagating in time, we rely on special “stop-gradient” operations. In the Appendix, we provide pseudocode and discussion of how this can be achieved.

3. EXPERIMENTS

3.1. Regression With Poisson Spike Trains

To illustrate the inner workings of DECOLLE, we first demonstrate DECOLLE in a regression task. A three-layer fully connected network consisting of 512 neurons each is stimulated with a fixed 500 ms Poisson spike train. Each layer in the network is trained with a different pseudo-target: \hat{Y}^1 , a ramp function;

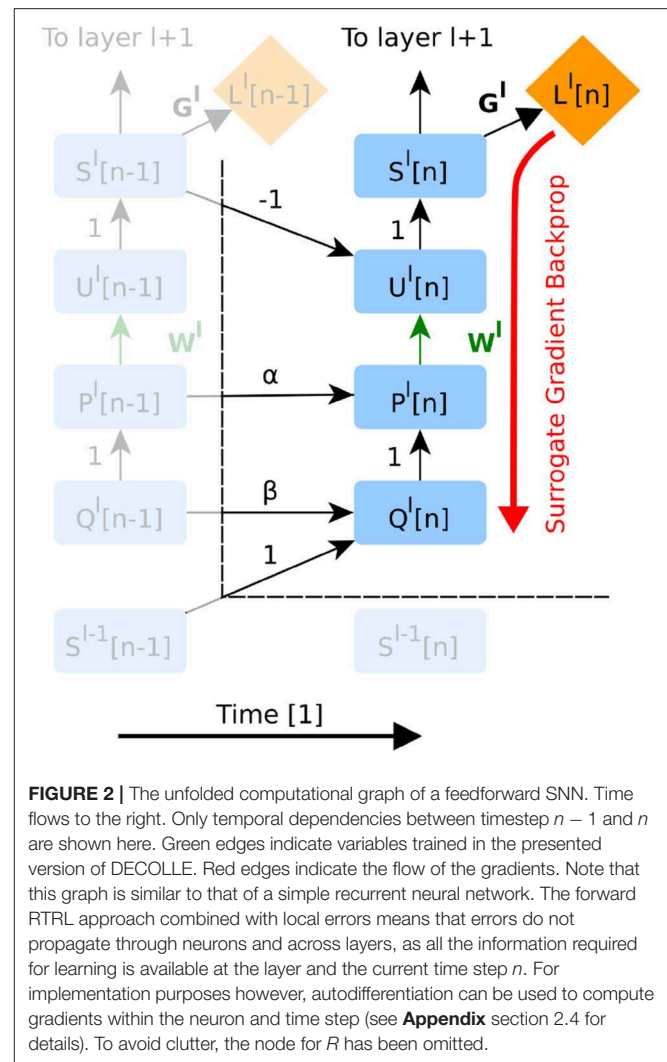


FIGURE 2 | The unfolded computational graph of a feedforward SNN. Time flows to the right. Only temporal dependencies between timestep $n - 1$ and n are shown here. Green edges indicate variables trained in the presented version of DECOLLE. Red edges indicate the flow of the gradients. Note that this graph is similar to that of a simple recurrent neural network. The forward RTRL approach combined with local errors means that errors do not propagate through neurons and across layers, as all the information required for learning is available at the layer and the current time step n . For implementation purposes however, autodifferentiation can be used to compute gradients within the neuron and time step (see Appendix section 2.4 for details). To avoid clutter, the node for R has been omitted.

\hat{Y}^2 , a high-frequency sinusoidal function and \hat{Y}^3 , a low-frequency sinusoidal function. Figure 1 illustrates the states of the neurons. For illustration purposes, the recording of the neural states was made in the absence of parameter updates (i.e., the learning rate is 0). The refractory mechanism decreases the membrane potential after the neuron spikes ($U[t]$). As discussed in the methods we use regularization on the membrane potential to keep the neurons from sustaining high firing rates and an activity regularizer to maintain a minimum firing rate. Updates to the weight are made at each time step and can be non-zero when the derivative of the activation function $\sigma'(U)$ and P are non-zero. The magnitude and direction of the update are determined by the error. Note that, in effect, the error is randomized as a consequence of the random local readout. The network learned to use the input spike times to reliably produce the targets.

3.2. N-MNIST

The N-MNIST dataset was recorded with a Dynamic Vision Sensor (DVS) (Lichtsteiner et al., 2008) mounted on a pan-tilt unit performing microsaccadic motions in front of a screen displaying samples from the MNIST dataset (Orchard et al.,

³Gradient BP is a special case of reverse mode AD, see Baydin et al. (2017) for complete review.

2015). Unlike standard imagers, the DVS records streams of events that signal the temporal intensity changes at each of its 128×128 pixels. For each of the 10 digits, we used 2,000 samples for training and 100 samples for testing. The samples are cropped spatially from 34×34 to 32×32 and temporally to 300 ms for both the train and test set. The network is simulated with a 1ms resolution—in other words, we sum up events in 1 ms time bins. No further pre-processing is applied to the events.

Events are separated in two channels with respect to their polarity. A N-MNIST sample is therefore represented as a tensor of shape $300 \times 2 \times 32 \times 32$, stacked into mini-batch of 500 samples. The DECOLLE network is fed with 1 ms slices of the input at a time. We relied on the same three-layer convolutional architecture used in the DvsGesture task described below. After a “burn-in” period of 50 ms during which no update is made,

gradient updates are performed at every simulation step. Hence, there are 250 weight updates per mini-batch. While the relevance of the time domain in N-MNIST is debatable (Iyer et al., 2018), this experiment shows that the neural dynamics of our network leads to successful classifications in under 300 ms.

The results on the N-MNIST dataset are shown in **Figure 3**. The experiment was performed 10 times with different random seeds. DECOLLE’s final error is $0.96 \pm 0.12\%$ for the third layer with 600,000 training iterations. We note that, due to the large memory requirements, it is not practical to train the DECOLLE convolutional network using BPTT. Hence we cannot provide BPTT baselines.

3.3. DvsGesture

We test DECOLLE at the more challenging task of learning gestures recorded using a DVS. Amir et al. recorded the DvsGesture dataset using a DVS, which comprises 1,342 instances of a set of 11 hand and arm gestures, collected from 29 subjects under three different lighting conditions (Amir et al., 2017). The unique features of each gesture are embedded in the stream of events. The event streams were downsized from 128×128 to 32×32 (events from four neighboring pixels were summed together as a common stream) and binned in frames of 1ms, the effective time step of the GPU-based simulation (**Figure 4**). During training, a sample consisted of 500 ms-long slices of the sample. To maximize the use of the dataset, the starting point of the slice was picked randomly, but such that a full 500 ms sequence could be constructed. The sequences were presented to the network in mini-batches of 72 samples. Testing sequences were 1,800 ms-long, each starting from the beginning of each recording (288 testing sequences). Note that since the shortest recording in the test set is 1,800 ms, this duration was selected to simplify and speed up the evaluation. The classification is obtained by counting spikes at the output starting from a burn-in period of 50 ms and selecting as output class the neuron that spiked the most. Test results from the DECOLLE network are reported with the dropout layer kept active, as this provided better results. Contrary to Amir et al. (2017), we did not use stochastic decay and the neural network structure

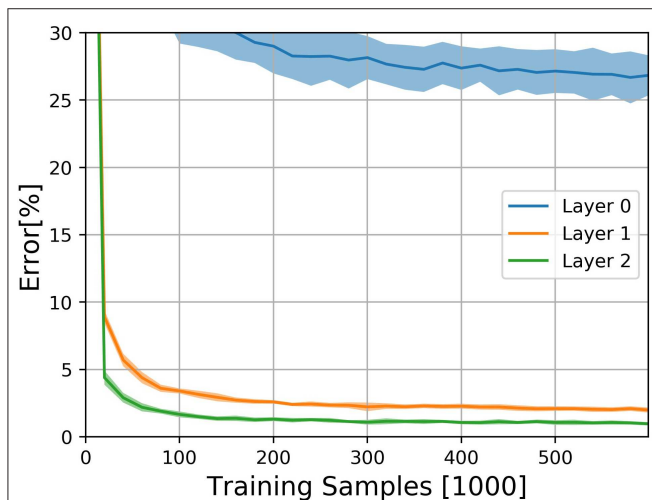


FIGURE 3 | Classification results on the N-MNIST dataset for the three DECOLLE layers. Classification Error for the N-MNIST task during learning for all local errors associated with the convolutional layers. Shadings indicate standard deviation across the 10 runs.

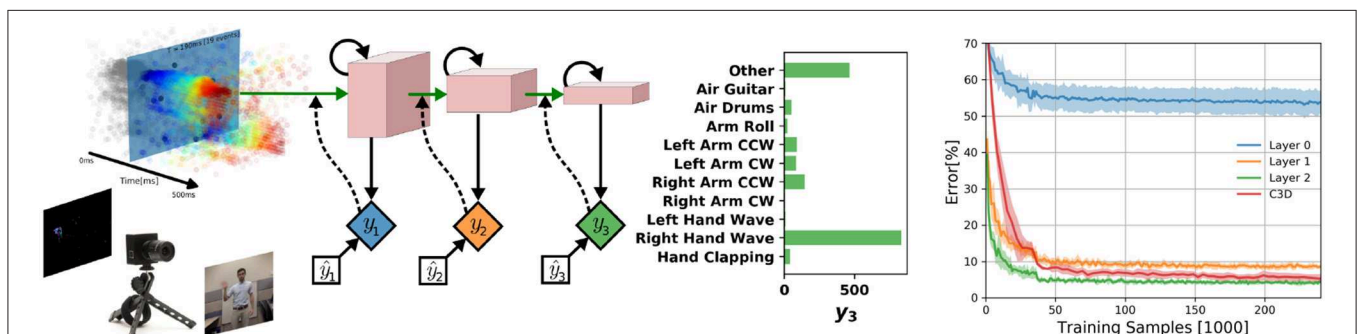


FIGURE 4 | (Left) DECOLLE setup for DvsGesture recognition. Learning was performed on the dataset provided with Amir et al. (2017) and consists of 11 gestures. The network consisted of three convolutional layers with max-pooling. A local classifier is attached to every layer and followed by dropout for regularization. DECOLLE is fed with 1 ms integer frames. **(Right)** Classification Error for the DvsGesture task during learning for all local errors associated with the convolutional layers. Shadings indicate standard deviation across runs (5 runs for C3D, 10 runs for DECOLLE).

TABLE 1 | Classification error at the DvsGesture task.

Model	Error	Training	Iterations	References
DECOLLE	4.46 ± 0.16%	Online	0.16M	This Work
SLAYER	6.36 ± 0.49 %	BPTT	0.27M	Shrestha and Orchard, 2018
C3D	5.46 ± 1.06%	BPTT	0.32M	Tran et al., 2015
IBM EEDN	8.23% (5.51%)	BPTT	64M	Amir et al., 2017

The IBM EEDN error in parentheses refers to the case with sliding window filter. Bold values emphasize that this work achieves the lowest error.

is a three-layer convolutional neural network, loosely adapted from Springenberg et al. (2014). We did not observe significant improvement by adding more than three convolutional layers. In shallow convolutional neural networks, it is common to use larger kernel sizes (LeCun et al., 1998; Kubilius et al., 2019) (Table 2). Since the input sizes were 32×32 , we used 7×7 kernel sizes in DECOLLE to ensure that the receptive field of neurons in the last layer covered the input. The optimal hyperparameters were found by a combination of manual and grid search. The learning rate was divided by 5 every 500 steps.

We compared with C3D and energy-efficient deep networks (EEDN). EEDN is a convolutional deep neural network architecture that can be trained offline (e.g., on a GPU) and deployed on the IBM TrueNorth chip (Esser et al., 2016). EEDN was applied to DVS gestures and provides an important benchmark on this task (Amir et al., 2017). Because EEDN was not designed to utilize the temporal dynamics of the spiking neurons in IBM TrueNorth chip, time is represented using the channel dimension of 2D convolutional networks. This approach limits the length of the sequence that EEDN can process. To overcome this, Amir et al. (2017) used a sliding window filter. C3D is a 3D convolutional network commonly used for spatiotemporal classification in videos (Tran et al., 2015), where the dimensions are time, height, and width. Using 3D kernels, C3C can learn spatiotemporal patterns. The network was similar to Tran et al. (2015) except that it was adapted for 32×32 frames and using half of the features per layer (see Appendix for network layers). We note that the C3D network is deeper and wider than the DECOLLE network. We found that $16 \times 32 \times 32$ frames, where each of the 16 representing 32ms slices of the DVS data performed best.

Overall, DECOLLE's performance is comparable or better than other published SNN implementations that use BP for training (Table 1, Figure 4) and close to much larger C3D networks. DECOLLE reached the reported accuracies after two orders of magnitude fewer iterations and smaller network compared to the IBM EEDN case (Table 1) (Amir et al., 2017).

Interestingly, the first layer of DECOLLE has a low classification accuracy. A similar effect is observed in non-spiking neural networks Mostafa et al. (2017). The local classifier errors improve for the second and third hidden layers compared to the first hidden layer. This is an indication that the network is able to make use of depth to obtain better accuracy. An examination of the filters learning in the first convolutional layer shows filters of varying frequencies and orientations (Figure 5). Interestingly,

TABLE 2 | DECOLLE Neural network used for the DvsGesture dataset.

Layer type	#	Data type	Dimensions
DVS	2	AEDAT 3.1	128×128
Downsample (Sum)	2	Integer	32×32
7×7 Conv	64	Float	30×30
2×2 MaxPool	64	Float	15×15
Spiking Non-linearity		Binary	
Dropout ($p = 0.5$)		Float	
Dense	11	Float	11
7×7 Conv	128	Float	13×13
Spiking Non-linearity		Binary	
Dropout ($p = 0.5$)		Binary	
Dense	11	Float	11
7×7 Conv	128	Float	11×11
2×2 MaxPool	128	Float	5×5
Spiking Non-linearity		Binary	
Dropout ($p = 0.5$)		Binary	
Dense	11	Float	11

Note that dense layers are used for the local classifiers only and were not fed to the subsequent convolutional layers. AEDAT 3.1 is a data format used for event-based data. The spiking nonlinearity was always applied after the pooling layers. Dropout layers were left active during testing.



FIGURE 5 | 7×7 Filters learned in the positive polarity channel (Left) and negative polarity channel (Right) of the first convolutional layer. The similarity of the kernels across the two polarities reflects the DVS data, where leading edges and trailing edges co-occur with opposite polarities.

the filters on the positive and negative channels of the DVS are similar, but exhibit small variations that are consistent with motion. This correlation is consistent with the DVS data, where leading edges of one polarity co-occur with trailing edges of opposite polarity.

4. CONCLUSION

Understanding and deriving neural and synaptic plasticity rules that can enable hidden weights to learn is an ongoing quest in neuroscience and neuromorphic engineering. From a machine learning perspective, locality, and differentiability are key issues of the spiking neuron model operations. While the latter problem is now being tackled with surrogate gradient approaches, how to achieve this in deep networks in a scalable and local fashion is still an open question.

We presented a novel synaptic plasticity rule, DECOLLE, derived from a surrogate gradient approach with linear scaling in the number of neurons. The rule draws on recent work in surrogate gradient descent in spiking neurons and local learning with layerwise classifiers. The linear scalability is obtained through a (instantaneous) rate-based cost function on the local classifier. The simplicity of the DECOLLE rule equation makes it amenable for direct exploitation of existing machine learning software libraries. Thanks to the surrogate gradient approach, the updates computed through automatic differentiation are equal to the DECOLLE update. This enables the leveraging of a wide variety of machine learning frameworks for implementing online learning of SNNs.

Updates in DECOLLE are performed at every time step, in accordance with the continuity of the leaky I&F dynamics. This can lead to a large number of updates and inefficient implementations in hardware. To tackle this problem, updates can be made in an error-triggered fashion, as discussed in Payvand et al. (2020). A direct consequence of the local classifiers is the lack of cross-layer adaptation of the layers. To tackle this problem, one could use meta-learning to adapt the random matrix in the classifier. In effect, the meta-learning loop would act as the outer loop in the synthetic gradients approach Jaderberg et al. (2016). The notion that a “layer” of neurons specialized in solving certain problems and sensory modalities is natural in computational neurosciences and can open multiple investigation avenues for understanding learning and plasticity in the brain.

DECOLLE is a departure from standard SNNs trained with Hebbian spike-timing-dependent plasticity, as it uses a normative learning rule that is partially derived from first principles. Models of this type can make use of standard processors where it makes the most sense (i.e., readout, cost functions etc.) and neuromorphic dedicated hardware for the rest. Because it

leverages the best of both worlds, DECOLLE is poised to make SNNs take off in event-based computer vision.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation, to any qualified researcher.

AUTHOR CONTRIBUTIONS

JK, HM, and EN wrote the paper and conceived the experiments. JK and EN ran the experiments and analyzed the data.

FUNDING

This manuscript has been released as a pre-print at <https://arxiv.org/abs/1811.10766> (Kaiser et al., 2018). EN was supported by the Intel Corporation, the National Science Foundation under grant 1652159, and by the Korean Institute of Science and Technology. JK was supported by a fellowship within the FITweltweit programme of the German Academic Exchange Service (DAAD). HM was supported by the Swiss National Fund. The authors declare that this study received funding from Intel Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fnins.2020.00424/full#supplementary-material>

REFERENCES

- Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., et al. (2017). “A low power, fully event-based gesture recognition system,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 7243–7252. doi: 10.1109/CVPR.2017.781
- Baldi, P., Sadowski, P., and Lu, Z. (2017). Learning in the machine: the symmetries of the deep learning channel. *Neural Netw.* 95, 110–133. doi: 10.1016/j.neunet.2017.08.008
- Bartolozzi, C., and Indiveri, G. (2006). “Silicon synaptic homeostasis,” in *Brain Inspired Cognitive Systems, BICS 2006* (Lesvos), 1–4.
- Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. (2018). “Assessing the scalability of biologically-motivated deep learning algorithms and architectures,” in *Advances in Neural Information Processing Systems* (Montréal, QC), 9368–9378.
- Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.* 18, 5595–5637.
- 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*.
- 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*.
- Bohte, S. M., Kok, J. N., and La Poutré, J. A. (2000). “Spikeprop: backpropagation for networks of spiking neurons,” in *ESANN* (Bruges), 419–424.
- Brader, J., 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
- Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.* 23, 349–398. doi: 10.1007/s10827-007-0038-6
- Chicca, E., Stefanini, F., and Indiveri, G. (2013). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proc. IEEE* 102, 1367–1388. doi: 10.1109/JPROC.2014.2313954
- Clopath, C., Büsing, L., Vasilaki, E., and Gerstner, W. (2010). Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nat. Neurosci.* 13, 344–352. doi: 10.1038/nn.2479
- Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv [Preprint]. arXiv:1602.02830*.
- Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Joshi, P., Lines, A., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

- Eliasmith, C., and Anderson, C. (2004). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. MIT Press.
- 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
- Gerstner, W., and Kistler, W. (2002). *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press. doi: 10.1017/CBO9780511815706
- Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press. doi: 10.1017/CBO9781107447615
- 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
- Huayaney, F. L. M., Nease, S., and Chicca, E. (2016). Learning in silicon beyond STDP: a neuromorphic implementation of multi-factor synaptic plasticity with calcium-based dynamics. *IEEE Trans. Circuits Syst. I* 63, 2189–2199. doi: 10.1109/TCSI.2016.2616169
- Huh, D., and Sejnowski, T. J. (2017). Gradient descent for spiking neural networks. *arXiv [Preprint]*. arXiv:1706.04698.
- Iyer, L. R., Chua, Y., and Li, H. (2018). Is neuromorphic mnist neuromorphic? Analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv [Preprint]*. arXiv:1807.01013.
- Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., and Kavukcuoglu, K. (2016). Decoupled neural interfaces using synthetic gradients. *arXiv [Preprint]*. arXiv:1608.05343.
- Jaeger, H. (2001). *The “echo state” approach to analysing and training recurrent neural networks-with an erratum note*. Technical Report, German National Research Center for Information Technology GMD, Bonn.
- Kaiser, J., Mostafa, H., and Neftci, E. (2018). Synaptic plasticity for deep continuous local learning. *arXiv [Preprint]*. arXiv:1812.10766.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv Preprint*. arXiv:1412.6980.
- Kubilius, J., Schrimpf, M., Hong, H., Majaj, N. J., Rajalingham, R., Issa, E. B., et al. (2019). Brain-like object recognition with high-performing shallow recurrent ANNs. *arXiv [Preprint]*. arXiv:1909.06161.
- 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, 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
- Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). An 128x128 120dB 15μs-latency temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337
- Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support learning in deep neural networks. *arXiv [Preprint]*. arXiv:1411.0247.
- Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nat. Commun.* 7:13276. doi: 10.1038/ncomms13276
- 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
- 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., Ramesh, V., and Cauwenberghs, G. (2017). Deep supervised learning using local errors. *arXiv [Preprint]*. arXiv:1711.06756. doi: 10.3389/fnins.2018.00608
- Neftci, E., Augustine, C., Paul, S., and Deterakis, G. (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324
- Neftci, E., Chicca, E., Indiveri, G., and Douglas, R. (2011). A systematic method for configuring VLSI networks of spiking neurons. *Neural Comput.* 23, 2457–2497. doi: 10.1162/NECO_a_00182
- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595
- 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
- Payvand, M., Fouda, M., Eltawil, A., Kurdahi, F., and Neftci, E. (2020). “Error-triggered three-factor learning dynamics for crossbar arrays,” in *2020 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)* (Genova). doi: 10.1109/AICAS48895.2020.9073998
- 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
- 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. doi: 10.1007/978-3-319-46493-0_32
- Shrestha, S. B., and Orchard, G. (2018). “Slayer: Spike layer error reassignment in time,” in *Advances in Neural Information Processing Systems* (Montréal, QC), 1412–1421.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: the all convolutional net. *arXiv [Preprint]*. arXiv:1412.6806.
- Sussillo, D., and Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron* 63, 544–557. doi: 10.1016/j.neuron.2009.07.018
- Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). “Learning spatiotemporal features with 3D convolutional networks,” in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago), 4489–4497. doi: 10.1109/ICCV.2015.510
- Urbanczik, R., and Senn, W. (2014). Learning by the dendritic prediction of somatic spiking. *Neuron* 81, 521–528. doi: 10.1016/j.neuron.2013.11.030
- Williams, R. J., and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* 1, 270–280. doi: 10.1162/neco.1989.1.2.270
- Zenke, F., and Ganguli, S. (2017). Superspike: Supervised learning in multi-layer spiking neural networks. *arXiv [Preprint]*. arXiv:1705.11146. doi: 10.1162/neco_a_01086

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.

Copyright © 2020 Kaiser, Mostafa and Neftci. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Neuromorphic Systems Design by Matching Inductive Biases to Hardware Constraints

Lorenz K. Muller*, Pascal Stark, Bert Jan Offrein and Stefan Abel

Neuromorphic Devices and Systems Group, Science and Technology Department, IBM Research Zurich, Rüschlikon, Switzerland

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

Martin Ziegler,
Technische Universität Ilmenau,
Germany
Thomas Pfeil,
Bosch Center for Artificial Intelligence,
Germany

*Correspondence:

Lorenz K. Muller
lorenz.muller@pm.me

Specialty section:

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

Received: 18 November 2019

Accepted: 09 April 2020

Published: 28 May 2020

Citation:

Muller LK, Stark P, Offrein BJ and
Abel S (2020) Neuromorphic Systems
Design by Matching Inductive Biases
to Hardware Constraints.
Front. Neurosci. 14:437.
doi: 10.3389/fnins.2020.00437

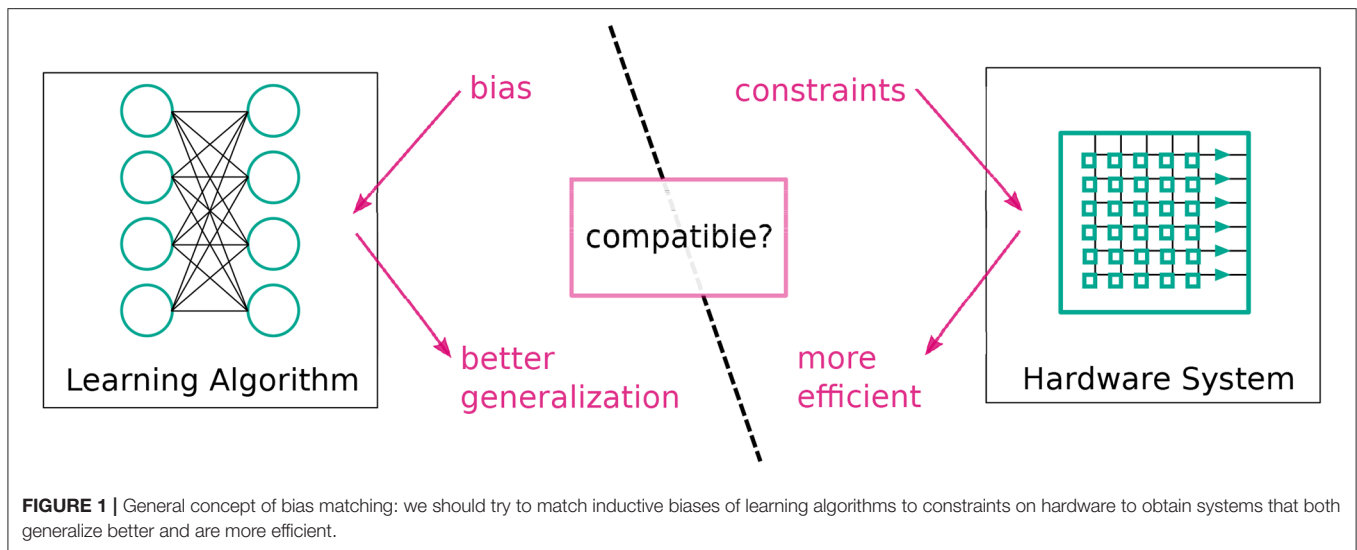
Neuromorphic systems are designed with careful consideration of the physical properties of the computational substrate they use. Neuromorphic engineers often exploit physical phenomena to directly implement a desired functionality, enabled by “the isomorphism between physical processes in different media” (Douglas et al., 1995). This bottom-up design methodology could be described as matching computational primitives to physical phenomena. In this paper, we propose a top-down counterpart to the bottom-up approach to neuromorphic design. Our top-down approach, termed “bias matching,” is to match the inductive biases required in a learning system to the hardware constraints of its implementation; a well-known example is enforcing translation equivariance in a neural network by tying weights (replacing vector-matrix multiplications with convolutions), which reduces memory requirements. We give numerous examples from the literature and explain how they can be understood from this perspective. Furthermore, we propose novel network designs based on this approach in the context of collaborative filtering. Our simulation results underline our central conclusions: additional hardware constraints can improve the predictions of a Machine Learning system, and understanding the inductive biases that underlie these performance gains can be useful in finding applications for a given constraint.

Keywords: neural network, neuromorphic, bias, constraint, inductive bias, sparsity, regularization, collaborative filtering

1. INTRODUCTION

A variety of systems are referred to as “neuromorphic.” Originally, “neuromorphic” has referred to the idea of making use of isomorphisms between physical processes in different media, for example, drift-diffusion phenomena in silicon to emulate drift-diffusion in neuronal ion channels, in order to build VLSI chips consisting of neuron-like elements (Mead, 1989; Douglas et al., 1995; Indiveri et al., 2011). Now, the term is used more broadly and also encompasses systems that accelerate artificial neural network (ANN) algorithms (Hu et al., 2016) or use a biomimetic processing principle (Furber, 2016).

Most neuromorphic systems have in common that parameters implemented in them or in the larger system around them are learned from examples. If this learning process should generalize to unseen examples, it is well-known that it needs to be biased in some way. Such biases that help a learning system generalize from its training data are known as inductive biases (Mitchell, 1980).



From an algorithmic perspective, inductive biases can come in many forms: the algorithm's structure, i.e., how parameters affect the output (Tai et al., 2015), regularization, e.g., additional costs (Krogh and Hertz, 1992), constraints on parameters during training (Ioffe and Szegedy, 2015), or, in the case of Bayesian models, explicitly as priors such as those described by Griffiths (2010). These concepts can also be interpreted as constraints on a learning algorithm's complexity. From this point of view, it is evident that, in an ideal world, the hardware on which the algorithm is implemented exploits this simplicity for more efficient processing (see **Figure 1**).

In this paper, we have asked two questions: "Can additional hardware constraints improve the predictions of a Machine Learning system?" and "What inductive biases underlie these performance gains?" We answered these questions by giving concrete examples and new conceptual designs of "bias matching": hardware implementations of machine learning algorithms, where a useful inductive bias can be exploited for efficient computation. Our aim was to establish "bias matching" as a high-level approach in the design of neuromorphic hardware.

"Bias matching" contrasts, as a design-philosophy, with a traditional bottom-up approach to neuromorphic engineering, e.g., as described by Douglas et al. (1995). In the bottom-up approach "the efficiency [...] rests in the power of analogy, the isomorphism between physical processes in different media" and "computational primitives such as conservation of charge, amplification, exponentiation, thresholding, compression and integration arise naturally out of the physical processes of aVLSI circuits." In this bottom-up approach the focus is on computational primitives and their efficient implementation, whereas we focus on inductive-biases and how to exploit them for efficiency.

We will discuss our approach based on examples concerning the following inductive biases/hardware constraints and elaborate why they may be relevant for hardware design (see also **Table 1** for an overview):

1. Translation and time-shift equivariance (section 2.1)
2. Spatio-temporal locality (section 2.2)
3. Frequency limitations of input signals (section 2.3)
4. Sparse, low-rank and kernelized low-rank connectivity (sections 2.4, 2.5 4.1)
5. Low-resolution connection weights (section 2.7)
6. Regularization by batch-size choice (sections 2.6, 4.2).

For each of these, we have defined an inductive bias or hardware constraint, given (where possible) an example of its relevance, and outlined how it can impact design. We have looked at hardware and software implementations from the literature (section 2), and we have presented novel observations and simulations to back up our claims (section 4).

2. BACKGROUND

In this section, we have examined examples of neural network implementations from the literature through the lens of the "bias matching" design perspective we propose in this paper.

2.1. Translation and Time-Shift Equivariance by Tying Weights

2.1.1. Inductive Bias

Probably the best known examples of an inductive bias in the context of neural networks are convolutional neural networks (CNNs) (Fukushima, 1988; LeCun et al., 1995) that exploit translation equivariance. Translation equivariance (e.g., Worrall et al., 2017), means that, if we translate the input of our model, its output remains the same up to a translation. On the example of images and CNNs, we can formulate translation equivariance: given a CNN layer $L(\cdot)$, image X , and the translation operator T , there exists an operator t such that

$$L(T(X)) = t(L(X)) \quad (1)$$

TABLE 1 | An overview of the pairings of hardware constraints and inductive biases discussed in this paper.

Hardware constraint	Inductive bias	Application area	Section
Tied weights between neighborhoods	Translation/Shift equivariance	Spatial and/or temporal signals, e.g., images, audio, video	2.1
Local communication/decaying memory	Local independence, spatially/temporally hierarchical models	Spatial and/or temporal signals, e.g., images, audio, video	2.2
Slow state change	Eigenvalues of recurrent network closer to one	Speech processing	2.3
Connectivity/memory limitations	Sparse and low-rank connectivity	Collaborative filtering, model compression	2.4, 4.1
Low-resolution weights	Difficult to interpret, possibly anti-synergistic with SGD	Unknown	2.7
No gradient aggregation over samples	Batch-Size regularization	Collaborative filtering (among others)	2.6, 4.2

Concretely, for a stride-1 convolution with appropriate padding, T and t are the same and $T(X_{i,j}) = t(X_{i,j}) = X_{i+t,j+s}$, where i and j are pixel indices, and s, t are small natural numbers. Notably, full CNNs are not translation equivariant, but single convolutional layers are.

CNNs achieve equivariance by enforcing that some of their parameters are equal (often referred to as “tying” parameters). A given neuron n in a CNN receives input from some window w of the previous layer’s output. For every other window w' there exists by construction a neuron n' with the same input weights as n (these weights are however applied to a different input, namely w'). In this sense, the weights of some sets of neurons in a CNN are tied together (not independent).

Note that time-shift equivariance is a special case of translation equivariance in one dimension.

2.1.2. Hardware Constraint

Because of its implementation in the form of weight tying, equivariance is highly relevant for hardware implementations of CNNs. All neurons belonging to the same input/output channel pair have the same weight. Hardware implementations of CNNs making use of this constraint, holding each weight in their memory only once and applying them to different sections of the input either by broadcasting (Bose et al., 2019) or sequentially, as, for example, on GPUs (Chetlur et al., 2014). On GPUs, the convolution operation is commonly recast as a highly optimized general matrix multiple between the filters and a copied and tiled input image (though many variants of GPU convolutions exist).

Recurrent neural networks (RNNs) keep their weights constant between subsequent time steps and implement time-shift equivariance in this way. Due to the sequential and non-linear nature of RNNs, implementations necessarily operate in a fixed time-unrolled order and explicitly implement weight tying.

2.1.3. Performance Impact

Weight-tied neural networks are state-of-the-art in many applications, particularly in the audio-visual domain. Some highly cited examples include digit recognition (LeCun et al., 1989), image classification (Cireřan et al., 2011; Krizhevsky et al., 2012; He et al., 2016), and speech recognition (Saon et al., 2017).

2.2. Spatial and Temporal Locality by Neighborhood Communication

2.2.1. Inductive Bias

Spatial locality means that, in order to compute a quantity of interest at a given point p , only points that lie in a small neighborhood of p need to be taken into account simultaneously. Spatially localized operations are often used to build a hierarchy of features with increasing spatial extent. The inductive bias associated with spatial locality is therefore either directly the independence of neighborhoods or the breaking down of concepts into a spatial hierarchy.

Temporal locality occurs commonly in Reservoir Computing (RC) (Lukořevičius et al., 2012) because most RC systems are designed such that the echo-state property (ESP) holds. Formally, the ESP states that the influence of any input signal vanishes asymptotically (Jaeger, 2007). RC is particularly effective when temporally local information is sufficient to solve the given task.

2.2.2. Hardware Constraint

Spatial locality is a key reason why CNNs can be computed efficiently (next to translation equivariance). The computation performed by neurons in a CNN is spatially localized if they have a small associated filter. An example of a hardware implementation of spatially localized processing is the SCAMP-5 sensor/processor array (Carey et al., 2013). The nearest-neighbor communication structure of this chip allows for an efficient pixel-parallel implementation of convolution filters if the filters are small (Bose et al., 2019). In GPU implementations of CNNs, small filters need fewer replications of each source pixel (as well as less memory for the filters themselves).

A class of hardware implementations that benefit from temporal locality are photonics-based RC, like Vandoorne et al. (2014). For silicon-photonics based systems, the integration of photonic amplifiers can be challenging, making temporal locality desirable. However, the operation of time-shifted addition (with small time shifts) is very efficient in these systems, allowing for cheap communication across “temporal neighborhoods.”

2.2.3. Performance Impact

For spatial locality and associated CNNs, see section 2.1.2.

An example of a very high through-put system made possible by temporal locality implemented in photonic hardware is Larger et al. (2017).

2.3. Low-Frequency Signal Components and Slow Neurons

2.3.1. Inductive Bias

For signals with slow dynamics (opposite to temporally local signals), an opposite approach can be useful. When analyzing signals, some of whose salient dynamics are much slower

than the sampling rate, it can be difficult to learn effective weights for recurrent neural networks (RNN) because longer time dependencies are more difficult to discover. A commonly used remedy for this is low-pass filtering of the hidden state of the RNN (Mozer, 1992). This inductive bias could also be described as enforcing eigenvalues of the recurrent connection matrix that are closer to one (Nair and Indiveri, 2019a) in a linearized approximation of the RNN.

2.3.2. Hardware Constraint

In a physical implementation, the fact that states of hidden neurons change slowly can be exploited by implementing them as leaky-integrate-and-fire (LIF) neurons with spike-frequency adaptation, which need to emit only few spikes to represent their state (Nair and Indiveri, 2019b). From the electrical engineering perspective, such neurons can be interpreted as $\Sigma\Delta$ -Modulators with unsigned Δ steps (Yoon, 2016).

2.3.3. Performance Impact

Nair and Indiveri (2019a,b) indeed observed that, when the time-constant of such neurons matches the salient structure of the analyzed signal (i.e., a favorable inductive bias in the sense of Mozer, 1992 is used), the resulting system exceeds the performance of an unconstrained system while operating at very low power.

2.4. Linear Low-Rank Matrix Approximation by Parameter Sharing

2.4.1. Inductive Bias

Strict low-rank matrix approximations (Koren et al., 2009) model a $n \times m$ matrix W as $W = QR$, where Q is $n \times k$ and R is $k \times m$, where k is the resulting rank of W . Equivalently we can write the entries of W as dot-products of rows and columns of Q and R respectively:

$$w_{ij} = \vec{q}_i \cdot \vec{r}_j. \quad (2)$$

Low-rank matrix approximations are commonly used to model very large matrices from sparse observations, for example, in collaborative filtering (Koren et al., 2009).

2.4.2. Hardware Constraint

Low-rank approximation of connection matrices in neural networks is straightforward to implement with efficiency gains on general matrix multipliers (GEMMs). This is because a connection matrix restricted to rank- k is equivalent to the interposition of a size- k layer with a linear activation function. In formulae, we can write for a neural network layer with an $n \times m$ weight matrix W that is rank- k ; it can be written as $W = QR$, where Q is $n \times k$ and R is $k \times m$:

$$W\vec{x} = QR\vec{x} = Q(R\vec{x}) = Q\vec{y}. \quad (3)$$

The time complexity of this moves from $O(mn)$ for $W\vec{x}$ to $O(k(m+n))$ for performing $R\vec{x}$ followed by $Q\vec{y}$. The memory required to store the parameters of W or R and Q respectively also scale this way: the individual entries of the matrix W share parameters.

2.4.3. Performance Impact

Low-rank reparameterization has been proposed as a model compression tool for neural networks, both fully-connected (Denil et al., 2013; Sainath et al., 2013) and convolutional ones (Jaderberg et al., 2014). Recent examples of practical efficiency tweaks that can be interpreted as low-rank approximation are the linear bottleneck and depth-wise convolutions of MobileNet-v2 (Sandler et al., 2018) (note that depth-wise convolutions may additionally interpose a non-linearity).

2.5. Kernelized Low-Rank Matrix Approximation by Parameter Sharing

2.5.1. Inductive Bias

Kernelized matrix reparameterization (Liu et al., 2016) generalizes the dot-product to any kernel function:

$$w_{ij} = K(\vec{q}_i, \vec{r}_j) \quad (4)$$

It has been shown that such kernelized reparameterizations can impose interpretable structure on neural networks (Muller et al., 2018).

2.5.2. Hardware Constraint and Performance Impact

Kernelized reparameterizations are more complicated to implement directly, and, to the best of our knowledge, this has not been discussed in the literature. The same reformulation as above does not work because the analog of \vec{y} would live in the embedding space of the kernel function, which can be infinitely dimensional. However, kernelized reparameterizations have greater representational power than strict low-rank approximations and have been shown to produce state-of-the-art results in collaborative filtering benchmarks (Muller et al., 2018). In the case of architectures where the limiting factor is memory access, kernelized reparameterizations can also be associated to a speed-up: instead of looking up the nm entries of W , they can be computed from only $k(n+m)$ values (entries of the matrix W share parameters). The additional overhead is the evaluation of a kernel function.

2.6. Batch-Size Regularization With Model Parallelism

2.6.1. Inductive Bias

Standard neural network training by stochastic gradient descent (SGD) and its variants can be seen as a kind of regularization or inductive bias in itself (Neyshabur et al., 2017) (SGD with a small learning rate, is more likely to find a solution with small parameter values). Furthermore, in the case of mini-batch gradient descent (where gradients are summed or averaged over a “mini-batch” of examples before being applied to weights), decreasing the batch-size is often associated with better generalization (Wilson and Martinez, 2003). This may, however, depend on the exact variant of gradient descent used (Smith, 2018).

2.6.2. Hardware Constraint and Performance Impact

For standard GPU implementations of neural networks, this is somewhat problematic because parallelization is most easily

implemented as data parallelism over the batch dimension (Chetlur et al., 2014; Krizhevsky, 2014). In the worst case, this results in a trade-off between speed-up and generalization performance. In contrast, hardware implementations with weight-wise parallelism in the vein of Gokmen and Vlasov (2016) can have difficulties aggregating gradients over multiple samples but do not have to make the speed-generalization trade-off.

2.7. Low Resolution Synaptic Weights

2.7.1. Inductive Bias

To the best of our knowledge, there is no clear inductive bias associated with the use of low-resolution synaptic weights, and, consequently, it is unclear what task or learning setup matches low resolution constraints. Intuitively, low resolution arithmetic might not match the setting of gradient-based training because the gradient only gives reliable information in a small neighborhood around the current model parameters. Circumstantial evidence for this is the significant amount of work on the improvement of training methods in the context of low-resolution weights (e.g., Müller et al., 2017; Alizadeh et al., 2019; Helwegen et al., 2019). More generally, Goodfellow et al. (2014) argue that current neural network architectures are selected under the constraint that they are well-suited for training by SGD. The improvement of alternative training methods (e.g., gradient-free ones) could, in light of this, be impactful for low resolution neural networks.

2.7.2. Hardware Constraint

In digital hardware, lower resolution directly translates into more compact designs. In analog hardware, there probably is an analogous trend due to noise tolerance. The optimal implementation of the low-resolution arithmetic for neural networks is in itself an open research question. Both floating-point (Courbariaux et al., 2014) and fixed-point (Lin et al., 2016) approaches exist combined with different number formats (Langroudi et al., 2019) and compression approaches (Aimar et al., 2018). For the extreme case of binary and ternary weights (Courbariaux et al., 2015; Muller and Indiveri, 2015), the multiplication between inputs and synaptic weights can also be simplified, as in Courbariaux et al. (2015) or by sparse versions thereof.

2.7.3. Performance Impact

State-of-the-art neural networks, in terms of pure predictive power, use at least 16-bit floating point arithmetic in all applications we are aware of. However, some ultra-low-resolution systems are highly competitive in terms of performance per power (Andri et al., 2016) or performance under limited memory usage (Uhlich et al., 2020).

3. METHODS

In this section, we have given implementation details of the simulations in the following section. We limited ourselves to dense, technical descriptions here and have given more context in the following section. All models were implemented in tensorflow (Abadi et al., 2016).

3.1. Bias Matching

In section 4, we have given two examples of how to apply bias matching in a concrete situation. Here, we have provided an abstract step-by-step description of bias matching.

1. Define a hardware property or constraint.
2. Define an end-to-end machine learning architecture incorporating the given constraint.
3. Find tasks that benefit from inductive biases associated with the constraint.

If necessary, revisit point two after the evaluating performances. While we followed this series of steps in the examples, one could also take an inductive bias as the starting point and work toward a hardware constraint.

3.2. Sparse Connectivity With Recurrent Fixed Weights

In this section, we have defined a neural network layer whose performance we have compared to that of a standard fully-connected layer in two different settings.

The layer we proposed, termed the sparseRec-layer, has the following recurrent definition (the reasoning behind this definition is given in section 4.1), given input \vec{x} :

$$\begin{aligned}\vec{y}_0 &= W_{in}\vec{x} \\ \vec{y}_t &= f(\vec{y}_0^* + W_{rec}\vec{y}_{t-1})\end{aligned}\quad (5)$$

where W_{in} is a learned input $n \times k$ matrix, and W_{rec} is a fixed, randomly drawn recurrent $m \times m$ connection matrix. We chose $m > k$ and will denote $s = \frac{k}{m}$ as sparsity. \vec{y}_0^* is \vec{y}_0 zero-padded from length k to m . $f(\cdot)$ is an activation function. When computing the output of such a layer, we applied this recurrent definition up to t_{max} while keeping the input fixed.

3.2.1. MNIST

The baseline model is a multilayer perceptron with one hidden layer trained on MNIST (LeCun et al., 1998). The hidden layer has $m \in \{16, 31, 62, 125, 250, 500\}$ neurons and a rectified-linear activation function (Glorot et al., 2011). We trained with the Adam optimizer (Kingma and Ba, 2014) for 40 epochs at a batch size of 256 and summed categorical cross-entropy cost. We used drop-out regularization (Srivastava, 2013). We ran a hyperparameter sweep for dropout values $d \in \{0.0, 0.2, 0.4, \text{ and } 0.6\}$ and learning rates $l \in \{0.0005, 0.001, 0.002, \text{ and } 0.003\}$ with five different random seeds. We selected the best performing parameters on a validation set and report the best average performance of each model.

Formulaically the networks prediction given input \vec{x} is

$$y = \text{softmax}(W_{out} \cdot \text{Dropout}(\text{ReLU}(W_{in} \cdot \text{Dropout}(\vec{x})))) \quad (6)$$

The sparseRec model is identical with some changes: the hidden layer is replaced with a sparseRec-layer, as described in Equation (5), and also has a rectified-linear activation function. The sparsity s and the corresponding number of non-zero columns k of the feedforward matrix W_{in} is given in **Table 2**. The values of W_{in} are constrained to lie in $[-1, 1]$ by reprojection after

TABLE 2 | Basic network parameters used in the simulations in section 3.2.

# hidden units (m)	# connected hidden units (k)	Sparsity (s) (%)
500	500	0
500	250	50
500	125	75
500	62	87.6
500	32	93.2
500	16	96.8

each optimization step. The recurrent matrix W_{rec} is set to fixed uniformly random weights of density 0.2 and rescaled to have spectral radius 0.95 (motivated by the echo-state property, this limits gradient decay/explosion).

Our goal was to compare the predictive accuracy of the two models as a function of the number of free parameters.

3.2.2. ML1M

The baseline model is an item-based autoencoder identical to the one described in Sedhain et al. (2015). It has one or two hidden layers with $m \in \{280, 300, 350, 400, 450, 500\}$ neurons (each) and sigmoid activation function. We optimized using full batches and the L-BFGS optimizer (Zhu et al., 1997) on the summed squared error of known entries with an L2 regularization strength $l_2 \in \{25, 50, 100\}$. The L2 regularization was applied to the connection weights (not to biases) in the form of a cost $c = l_2 \sum_{ij} (W_{ij})^2$.

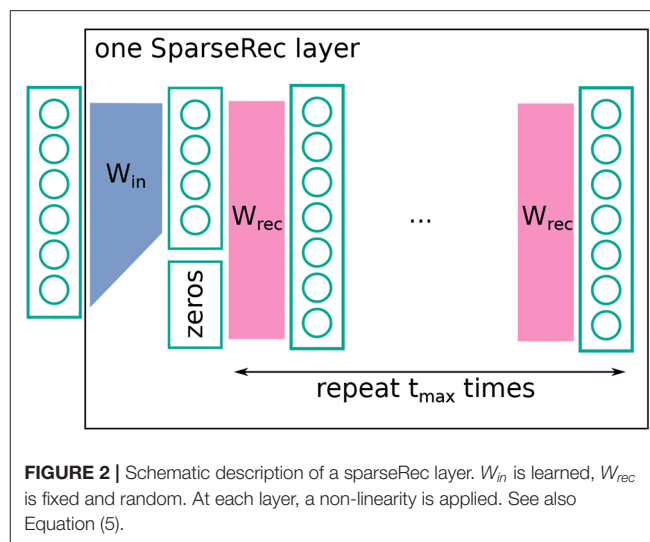
The sparseRec model is identical with the some changes: the hidden layer is replaced with a sparseRec-layer, as described in Equation (5), with a sigmoid activation function (as in the baseline model). The sparsity s and the corresponding number of non-zero columns k of the feedforward matrix W_{in} is given in Table 2. The values of W_{in} are constrained to lie in $[-1, 1]$ by reprojection during the optimization. The recurrent matrix W_{rec} was set to fixed uniformly random weights of density 0.2 and rescaled to have spectral radius 0.95.

As for the MNIST dataset, we compared the predictive accuracy of the two models.

3.3. Batch-Size Regularization in Low-Rank Matrix Approximation

The model used was an Factorization Machine (FM) as described in Rendle (2012), where we adopted two minor deviations from this description also used in the code accompanying that paper: weight-decay (L2 regularization) was only applied to parameters that have non-zero gradient, and the models output was restricted to the range of the rating values given in the training set. Finally, we added a modification for numerical stability with large batch sizes: the gradient of the global bias b was divided by the batch-size.

For each batch-size, we individually found the optimal hyperparameters (L2 regularization strength l_2 , learning rate) in $\{0.02, 0.04, 0.06, 0.08\} \times \{0.0005, 0.001, 0.002, \text{ and } 0.003\}$. l_2 is the multiplicative coefficient to an L2 cost given in the previous subsection. We ran each model for at least five different random



seeds (resulting in different initial parameters and different train-test splits). For each batch-size, we picked the hyperparameters with the best average performance.

Our goal was to examine the test accuracy of the model as a function of the training batch size.

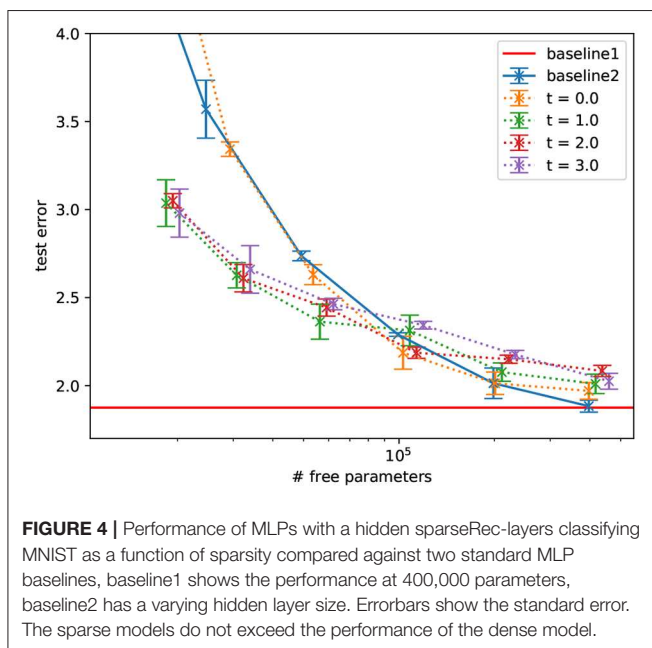
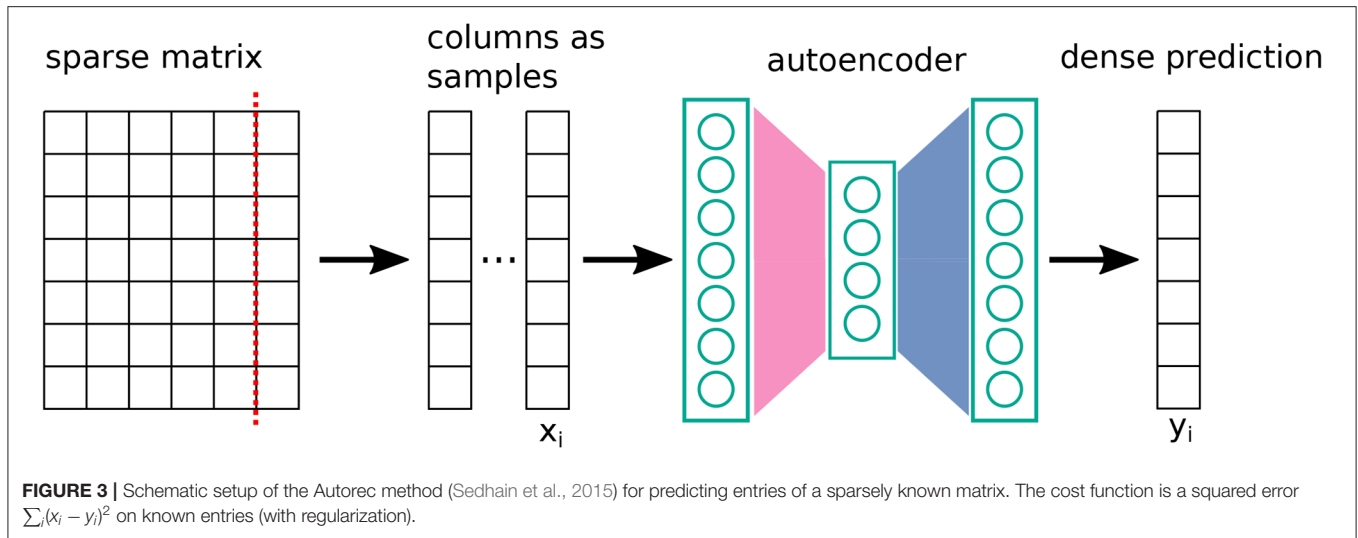
4. SIMULATION RESULTS AND DISCUSSION

In this section, we have shown simulation results where we could identify good use-cases for specific computational limitations. In these use-cases, the limitations match a task's preferred inductive bias. We further observed that, for other tasks, the same biases may well lead to a deterioration in performance. We emphasize that we did not perform exhaustive architecture searches for a given task but conversely performed a constraint search for an architecture and application that leads to an improvement over a baseline.

4.1. Sparse Connectivity With Fixed Weights

In this subsection, we began from a particular hardware constraint and tried to find a suitable application for it, following section 3.1 (step 1): we assumed we had developed hardware that would allow us to cheaply multiply vectors with a fixed, uniformly random matrix.

Next, we defined an architecture (step two in section 3.1). The architecture we considered could be succinctly described as a deep echo-state network (ESN) (Gallicchio and Micheli, 2016) with trained feed-forward weights or alternatively as a set of sparsely connected feed-forward layers, with fixed random recurrent connections within each layer (see Figure 2 for a visual explanation). As given in the previous section, formulaically we proposed a neural network layer, termed the sparseRec-layer,

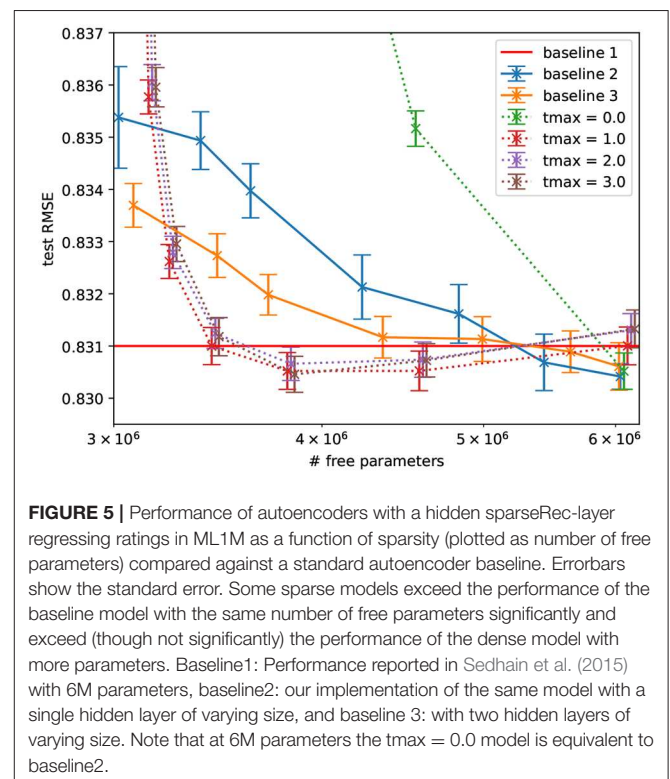


with the following recurrent definition, given input \vec{x} :

$$\begin{aligned}\vec{y}_0 &= W_{in}\vec{x} \\ \vec{y}_t &= f(\vec{y}_0^* + W_{rec}\vec{y}_{t-1})\end{aligned}$$

where W_{in} is a learned input $n \times k$ matrix, and W_{rec} is a fixed, randomly drawn recurrent $m \times m$ connection matrix. We chose $m > k$ and denoted $s = \frac{k}{m}$ as sparsity. \vec{y}_0^* is \vec{y}_0 zero-padded from length k to m . $f(\cdot)$ was an activation function. When computing the output of such a layer, we applied this recurrent definition up to t_{max} while keeping the input fixed.

The intuition behind a sparseRec-layer, is that we want a high number of linearly independent activations in each layer. Simultaneously, we wanted to keep the number of adjustable weights small (for regularization and simpler hardware



implementation) compared to the number of fixed weights. From an implementation perspective, this architecture is interesting for some of the reasons that also make ESN and Extreme learning machines (ELM) (Huang et al., 2004) appealing to hardware designers: the use of mostly fixed weights (that do not need an updating mechanism) and the recurrent network structure (that reduces information transport in comparison to a feed-forward structure). Since we added trained feed-forward weights, we required that a product of an error vector with the transpose weight matrix could also be performed for the purpose of error back-propagation (in contrast to a

standard deep-esn). Crossbar-arrays (Steinbuch, 1961) are a well-known example of a kind of architecture that can support such operations.

As a “naive” first benchmark, we used MNIST (LeCun et al., 1998) and compared fully-connected networks to the proposed sparse networks with fixed random recurrent weights in each layer, as a function of number of free parameters (see **Figure 4**). We found a gradual degradation of the performance as the number of free parameters decreases. This is not surprising: we are not aware of any reason to expect that sparsity should improve performance for this task. Indeed, the sparse models do not exceed the performance of the dense model.

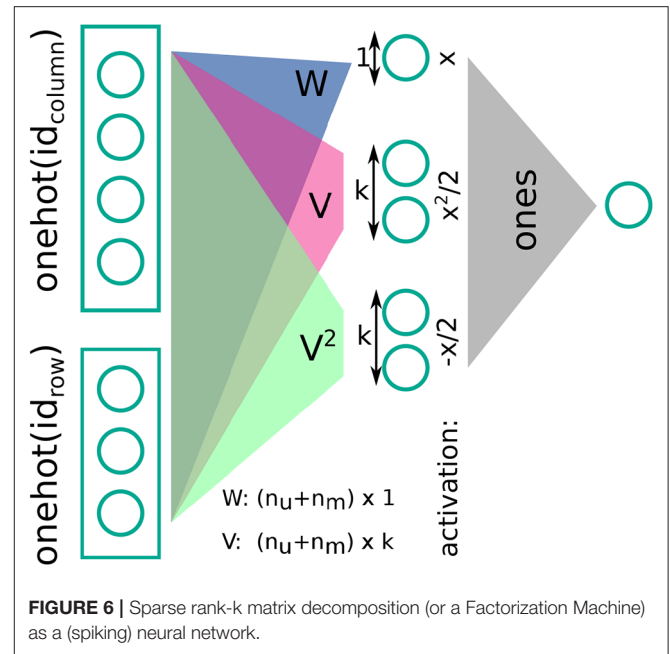
In contrast, it has been observed that sparsified networks can show improved performance in collaborative filtering settings (Muller et al., 2018) (step 3 in section 3.1). In the spirit of bias matching, we investigated whether our given sparse architecture would improve over the fully-connected baseline in this task. The setup followed (Sedhain et al., 2015) (see **Figure 3**). The goal was to regress missing entries of a large, sparsely known matrix given in MovieLens-1M (Harper and Konstan, 2015). To achieve this, the matrix was cut into columns or rows. Each column was treated as a sample. An autoencoder was trained to reconstruct columns, where the cost is the squared error for known entries and zero otherwise, in combination with L2-regularization. Training was performed by a gradient-descent variant, namely, L-BFGS (Zhu et al., 1997). We used the same network of Sedhain et al. (2015) as a baseline, and, for comparison, we replaced the hidden layer with the layer given in Equation (5).

Figure 5 shows that, this collaborative filtering setting, the constraint that degraded performance for the MNIST dataset, improves the performance over the fully connected baseline at a given number of free parameters. The performance also does not significantly change when decreasing the number of parameters by sparsifying in the proposed way but decreases significantly when the hidden layer is made smaller (to reach the same number of free parameters). We further found that additional network depth explains this in part by comparison to an architecture with an equal number of parameters and two hidden layers. Overall, this suggests that the constraint matches well the inductive bias required to generalize on this task.

Furthermore, we found that applying the fixed, random matrix more than once does not improve the performance significantly ($t_{\max} = 1$ is as good as $t_{\max} > 1$). This means that our final layer architecture could be described as a learned input matrix, followed by a fixed, random matrix; in spirit, this is closer to an ELM than an ESN.

4.2. Batch-Size Regularization in Low-Rank Matrix Approximation

As a second example (step 1 in section 3.1), we considered a “sparse vector”-“dense matrix” multiplier where the input data vector is binary, $\vec{x} \in \{0, 1\}^n$, and changes to matrix entries must occur in place. An example of such a system would be a spiking neural network with synapses implemented by a cross-bar array in the vein of Gokmen and Vlasov (2016). The key constraint we considered here is that such systems usually have difficulties aggregating gradients over multiple samples



(parallelization occurs across the weight-array instead of across mini-batches).

As a computational architecture (step 2 in section 3.1), we chose the Factorization Machine (FM) (Rendle, 2010). Given a sparse sample of the entries of a matrix, we wished to regress unknown entries. As a formula, the prediction for an entry r of the matrix, given the concatenated one-hot encoded row and column indices x , is

$$r = b + \sum_i x_i w_i + 0.5 \sum_j \left(\left(\sum_i v_{ij} x_i \right)^2 - \sum_i v_{ij}^2 x_i^2 \right). \quad (7)$$

From a neural network perspective, we can describe the setup as follows (see **Figure 6** for further details): we gave as input to three fully-connected layers the one-hot encoded row and column indices as a concatenated vector; two of these layers have a linear, the other a quadratic activation function, and their weights W, V_1, V_2 are tied such that $V_1 = V_2^2$. The weights have sizes $W: (n_c + n_r) \times 1$ and $V: (n_c + n_r) \times k$, where n_c, n_r are the number of columns and rows, respectively. The three hidden layers are read out by a dense layer of size one with fixed weights of plus one (i.e., they are summed). The output of this dense layer is the prediction for the rating. Training is performed by gradient-descent with L2-regularization.

We further note that **Figure 6** makes it evident that FMs are closely related to spiking neural networks in the sense that their central operation is a sparse vector-matrix multiplication. In addition, $\vec{x} \in \{0, 1\}^n$ has a clear interpretation for FMs: In this case, the FM solves a low-rank factorization problem (Rendle, 2010).

A key area of application of FMs are collaborative filtering tasks. We therefore considered low-rank matrix factorization of MovieLens-1M as a test application with the inductive bias of small training batch sizes (step 3 in section 3.1).

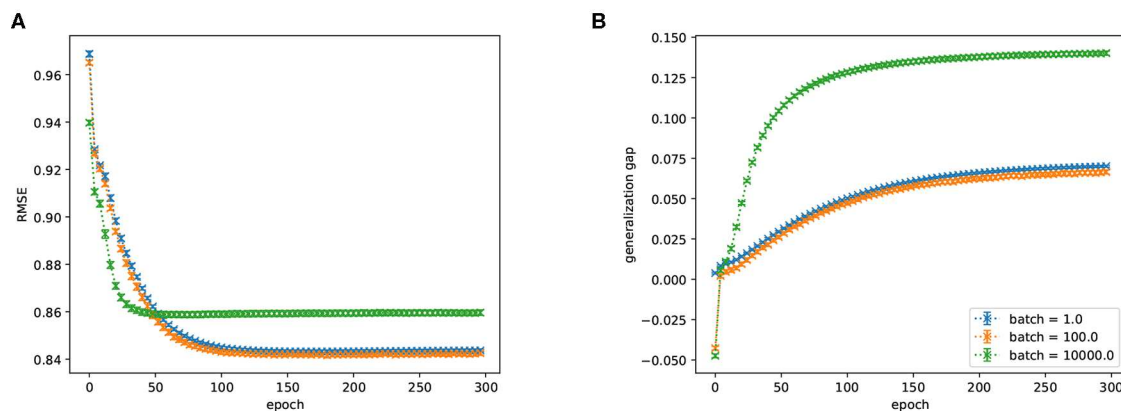


FIGURE 7 | (A) RMSE (lower is better) and **(B)** generalization gap (difference between train and test performance, lower is better) of a FM trained on ML1M as a function of training epoch for various batch sizes. There is a significant preference for smaller batch sizes. Batch-sizes 1 and 100 are not significantly different.

We plotted the performance on a validation set of this network as a function of the mini-batch size during training (**Figure 7**). We found that increasing batch-sizes reduce the performance of the network (but note that it is possible that means of regularization other than the ones we tested allow for the use of larger batch-sizes). This indicates that an interesting area of application for weight-parallel spiking neural network accelerators are FMs because they can give a (weight-wise) parallelization speed-up without the performance degradation associated with large batch-sizes.

We note that the beneficial effect of using SGD with small batch-sizes has been observed in other applications as well (as mentioned in section 2.6, e.g., Wilson and Martinez, 2003).

5. CONCLUSIONS

When one approximates a machine learning model efficiently, assuming some hardware constraints, the usefulness of these constraints for generalization is worth careful consideration. In other words, hardware constraints must match inductive biases. Such a match can lead to highly efficient and well-performing systems. For example, when designing a neuromorphic chip to analyze speech signals, it does not need to support fast state changes in the hidden neurons (see section 2.3), and building accelerators for collaborative filtering exploiting sparsity could be very relevant (see section 4.1).

Similarly, avoidance of an inappropriate bias can also be crucial, as demonstrated by the Shuffle-Net (Zhang et al., 2018), where a factorization of the model into independent subnetworks is avoided by random shuffling of sparsely connected channels.

Recently, the question has arisen as to whether, in machine learning research, the most successful approach is to look for ways to apply more computational power to a problem rather than finding better designed solutions (Sutton, 2019). Through the many examples of “bias-matching” we have reported in this paper, we support the contrary notion that finding low-level

improvements (through hardware constraints) that synergize with the problems one is trying to solve (through inductive biases) is a kind of thoughtful problem solving that can be crucial in the development of competitive machine learning systems.

The embodiment of inductive biases as hardware constraints also implies a caveat for the evaluation of neuromorphic architectures: if an architecture aims to be general purpose, it is important to benchmark it on a variety of tasks; otherwise, it may be the case that the chosen benchmarks benefit from inductive biases embodied by the constraints of the given architecture.

In this paper, we discussed several examples from the literature where such a match is given. Furthermore we applied the idea of bias matching to a novel network architecture that can make use of fixed, random weights, and found that its sparse structure leads to improved performance over a dense baseline on a benchmark for which sparsity has been shown to be useful previously.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

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

FUNDING

This study was partially funded through the Swiss National Science Foundation grant no. 175801 Novel Architectures for Photonic Reservoir Computing.

ACKNOWLEDGMENTS

LM would like to thank Julien N.P. Martel for helpful discussions.

REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.
- 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
- Alizadeh, M., Fernández-Marqués, J., Lane, N. D., and Gal, Y. (2019). “A systematic study of binary neural networks’ optimisation,” in *International Conference on Learning Representations*. Available online at: <https://openreview.net/forum?id=rJfUCoR5KX>
- Andri, R., Cavigelli, L., Rossi, D., and Benini, L. (2016). “Yodann: an ultra-low power convolutional neural network accelerator based on binary weights,” in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (Pittsburgh, PA: IEEE), 236–241. doi: 10.1109/ISVLSI.2016.111
- Bose, L., Chen, J., Carey, S. J., Dudek, P., and Mayol-Cuevas, W. (2019). “A camera that CNNs: towards embedded neural networks on pixel processor arrays,” in *Proceedings of the IEEE International Conference on Computer Vision* (Seoul), 1335–1344. doi: 10.1109/ICCV.2019.00142
- Carey, S. J., Lopich, A., Barr, D. R., Wang, B., and Dudek, P. (2013). “A 100,000 fps vision sensor with embedded 535GOPS/W 256 × 256 SIMD processor array,” in *2013 Symposium on VLSI Circuits* (Kyoto: IEEE), C182–C183.
- Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., et al. (2014). cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.
- Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2011). “A committee of neural networks for traffic sign classification,” in *2011 International Joint Conference on Neural Networks* (San Jose, CA: IEEE), 1918–1921. doi: 10.1109/IJCNN.2011.6033458
- Courbariaux, M., Bengio, Y., and David, J.-P. (2014). Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*.
- 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*, eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Montreal, QC: Curran Associates, Inc.), 3123–3131.
- Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and De Freitas, N. (2013). “Predicting parameters in deep learning,” in *Advances in Neural Information Processing Systems*, eds C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Curran Associates, Inc.) 2148–2156.
- Douglas, R., Mahowald, M., and Mead, C. (1995). Neuromorphic analogue VLSI. *Annu. Rev. Neurosci.* 18, 255–281. doi: 10.1146/annurev.ne.18.030195.001351
- Fukushima, K. (1988). Neocognitron: a hierarchical neural network capable of visual pattern recognition. *Neural Netw.* 1, 119–130. doi: 10.1016/0893-6080(88)90014-7
- Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001
- Gallicchio, C., and Micheli, A. (2016). “Deep reservoir computing: a critical analysis,” in *ESANN* (Bruges). doi: 10.1016/j.neucom.2016.12.089
- Glorot, X., Bordes, A., and Bengio, Y. (2011). “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (Ft. Lauderdale, FL), 315–323.
- 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
- Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2014). Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*.
- Griffiths, T. L. (2010). “Bayesian models as tools for exploring inductive biases,” in *Generalization of Knowledge: Multidisciplinary Perspectives*, eds M. T. Banich and D. Caccamise (Psychology Press), 135–156. doi: 10.21236/ADA566965
- Harper, F. M., and Konstan, J. A. (2015). The MovieLens datasets: history and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5:19. doi: 10.1145/2827872
- 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. doi: 10.1109/CVPR.2016.90
- Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.-T., and Nusselder, R. (2019). “Latent weights do not exist: Rethinking binarized neural network optimization,” in *Advances in Neural Information Processing Systems*, eds H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett (Vancouver, BC: Curran Associates, Inc.) 7531–7542.
- Hu, M., Strachan, J. P., Li, Z., Grafals, E. M., Davila, N., Graves, C., et al. (2016). “Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication,” in *Proceedings of the 53rd Annual Design Automation Conference* (Austin, TX: ACM), 19. doi: 10.1145/2897937.2898010
- Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. *Neural Netw.* 2, 985–990. doi: 10.1109/IJCNN.2004.1380068
- 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
- Ioffe, S., and Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*. doi: 10.5244/C.28.88
- Jaeger, H. (2007). Echo state network. *Scholarpedia* 2:2330. doi: 10.4249/scholarpedia.2330
- Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer* 8, 30–37. doi: 10.1109/MC.2009.263
- Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, eds F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc.), 1097–1105.
- Krogh, A., and Hertz, J. A. (1992). “A simple weight decay can improve generalization,” in *Advances in Neural Information Processing Systems*, 950–957.
- Langroudi, H. F., Carmichael, Z., and Kudithipudi, D. (2019). Deep learning training on the edge with low-precision posits. *arXiv preprint arXiv:1907.13216*.
- Larger, L., Baylón-Fuentes, A., Martinenghi, R., Udaltsov, V. S., Chembo, Y. K., and Jacquot, M. (2017). High-speed photonic reservoir computing using a time-delay-based architecture: million words per second classification. *Phys. Rev. X* 7:011015. doi: 10.1103/PhysRevX.7.011015
- LeCun, Y., and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* 3361:1995.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). *Neural Comput.* 1, 541–551. doi: 10.1162/neco.1989.1.4.541
- LeCun, Y., Cortes, C., and Burges, C. J. (1998). *The Mnist Database of Handwritten Digits*. Available online at: <http://yann.lecun.com/exdb/mnist> (accessed December 20, 2019).
- Lin, D., Talathi, S., and Annapureddy, S. (2016). “Fixed point quantization of deep convolutional networks,” in *International Conference on Machine Learning* (New York, NY), 2849–2858.
- Liu, X., Aggarwal, C., Li, Y.-F., Kong, X., Sun, X., and Sathe, S. (2016). “Kernelized matrix factorization for collaborative filtering,” in *Proceedings of the 2016 SIAM International Conference on Data Mining* (Miami, FL: SIAM), 378–386. doi: 10.1137/1.9781611974348.43
- Lukoševičius, M., Jaeger, H., and Schrauwen, B. (2012). Reservoir computing trends. *KI-Kunstliche Intelligenz* 26, 365–371. doi: 10.1007/s13218-012-0204-5
- Mead, C. (1989). *Analog VLSI and Neural Systems*. NASA STI/Recon Technical Report A, 90.
- Mitchell, T. (1980). *The Need for Biases in Learning Generalizations*. Rutgers computer science tech. rept. cbm-tr-117. Rutgers University.
- Moser, M. C. (1992). “Induction of multiscale temporal structure,” in *Advances in Neural Information Processing Systems*, 275–282.
- Muller, L. K., and Indiveri, G. (2015). Rounding methods for neural networks with low resolution synaptic weights. *arXiv preprint arXiv:1504.05767*.
- Muller, L. K., Martel, J., and Indiveri, G. (2018). “Kernelized synaptic weight matrices,” in *International Conference on Machine Learning* (Stockholm), 3651–3660.

- Müller, L. K., Nair, M. V., and Indiveri, G. (2017). "Randomized unregulated step descent for limited precision synaptic elements," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD: IEEE), 1–4. doi: 10.1109/ISCAS.2017.8050217
- Nair, M. V., and Indiveri, G. (2019a). A neuromorphic boost to RNNs using low pass filters. *arXiv preprint arXiv:1905.10692*.
- Nair, M. V., and Indiveri, G. (2019b). "An ultra-low power sigma-delta neuron circuit," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* (Sapporo: IEEE), 1–5. doi: 10.1109/ISCAS.2019.8702500
- Neyshabur, B., Tomioka, R., Salakhutdinov, R., and Srebro, N. (2017). Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*.
- Rendle, S. (2010). "Factorization machines," in *2010 IEEE International Conference on Data Mining* (Sydney, NSW: IEEE), 995–1000. doi: 10.1109/ICDM.2010.127
- Rendle, S. (2012). Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.* 3:57. doi: 10.1145/2168752.2168771
- Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013). "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (Vancouver, BC: IEEE), 6655–6659. doi: 10.1109/ICASSP.2013.6638949
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). "Mobilenetv2: inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 4510–4520. doi: 10.1109/CVPR.2018.00474
- Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., et al. (2017). English conversational telephone speech recognition by humans and machines. *arXiv preprint arXiv:1703.02136*. doi: 10.21437/Interspeech.2017-405
- Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. (2015). "Autorec: autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web* (Florence: ACM), 111–112. doi: 10.1145/2740908.2742726
- Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1-learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.
- Srivastava, N. (2013). Improving neural networks with dropout. *Univers. Toronto* 182:7.
- Steinbuch, K. (1961). Die lernmatrix. *Biol. Cybernet.* 1, 36–45. doi: 10.1007/BF00293853
- Sutton, R. S. (2019). *The Bitter Lesson*. Available online at: <http://www.incompleteideas.net/IncIdeas/BitterLesson.html> (accessed December 20, 2019).
- Tai, C., Xiao, T., Zhang, Y., Wang, X., Weinan, E. (2015). Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*.
- Uhlich, S., Mauch, L., Cardinaux, F., Yoshiyama, K., Garcia, J. A., Tiedemann, S., et al. (2020). "Mixed precision DNNs: all you need is a good parametrization," in *International Conference on Learning Representations* (Addis Ababa).
- Vandoorne, K., Mechet, P., Van Vaerenbergh, T., Fiers, M., Morthier, G., Verstraeten, D., et al. (2014). Experimental demonstration of reservoir computing on a silicon photonics chip. *Nat. Commun.* 5:3541. doi: 10.1038/ncomms4541
- Wilson, D. R., and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Netw.* 16, 1429–1451. doi: 10.1016/S0893-6080(03)00138-2
- Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. (2017). "Harmonic networks: deep translation and rotation equivariance," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 5028–5037. doi: 10.1109/CVPR.2017.758
- Yoon, Y. C. (2016). LIF and simplified SRM neurons encode signals into spikes via a form of asynchronous pulse sigma-delta modulation. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 1192–1205. doi: 10.1109/TNNLS.2016.2526029
- Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). "Shufflenet: an extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 6848–6856. doi: 10.1109/CVPR.2018.00716
- Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.* 23, 550–560. doi: 10.1145/279232.279236

Conflict of Interest: All authors were employed by the company IBM.

Copyright © 2020 Muller, Stark, Offrein and Abel. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Supervised Learning in All FeFET-Based Spiking Neural Network: Opportunities and Challenges

Sourav Dutta^{1*}, Clemens Schafer², Jorge Gomez¹, Kai Ni³, Siddharth Joshi² and Suman Datta¹

¹ Department of Electrical Engineering, College of Engineering, University of Notre Dame, Notre Dame, IN, United States,

² Department of Computer Science and Engineering, College of Engineering, University of Notre Dame, Notre Dame, IN,

United States, ³ Department of Microsystems Engineering, Rochester Institute of Technology, Rochester, NY, United States

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

Guoqi Li,
Tsinghua University, China
Lyes Khacef,
Université Côte d'Azur, France

*Correspondence:

Sourav Dutta
sdutta4@nd.edu

Specialty section:

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

Received: 16 January 2020

Accepted: 22 May 2020

Published: 24 June 2020

Citation:

Dutta S, Schafer C, Gomez J,
Ni K, Joshi S and Datta S (2020)
Supervised Learning in All
FeFET-Based Spiking Neural Network:
Opportunities and Challenges.
Front. Neurosci. 14:634.
doi: 10.3389/fnins.2020.00634

The two possible pathways toward artificial intelligence (AI)—(i) neuroscience-oriented neuromorphic computing [like spiking neural network (SNN)] and (ii) computer science driven machine learning (like deep learning) differ widely in their fundamental formalism and coding schemes (Pei et al., 2019). Deviating from traditional deep learning approach of relying on neuronal models with static nonlinearities, SNNs attempt to capture brain-like features like computation using spikes. This holds the promise of improving the energy efficiency of the computing platforms. In order to achieve a much higher areal and energy efficiency compared to today's hardware implementation of SNN, we need to go beyond the traditional route of relying on CMOS-based digital or mixed-signal neuronal circuits and segregation of computation and memory under the von Neumann architecture. Recently, ferroelectric field-effect transistors (FeFETs) are being explored as a promising alternative for building neuromorphic hardware by utilizing their non-volatile nature and rich polarization switching dynamics. In this work, we propose an all FeFET-based SNN hardware that allows low-power spike-based information processing and co-localized memory and computing (a.k.a. in-memory computing). We experimentally demonstrate the essential neuronal and synaptic dynamics in a 28 nm high-K metal gate FeFET technology. Furthermore, drawing inspiration from the traditional machine learning approach of optimizing a cost function to adjust the synaptic weights, we implement a surrogate gradient (SG) learning algorithm on our SNN platform that allows us to perform supervised learning on MNIST dataset. As such, we provide a pathway toward building energy-efficient neuromorphic hardware that can support traditional machine learning algorithms. Finally, we undertake synergistic device-algorithm co-design by accounting for the impacts of device-level variation (stochasticity) and limited bit precision of on-chip synaptic weights (available analog states) on the classification accuracy.

Keywords: neuromorphic computing, supervised learning, surrogate gradient learning, ferroelectric FET, spiking neural network, spiking neuron, analog synapse

INTRODUCTION

Machine learning, especially deep learning has been a *de facto* choice for solving a wide range of real-world complex tasks and has contributed to the unprecedented success story of artificial intelligence (AI) in recent years. Fueled by large datasets and high-performance processors like GPU and TPU, deep learning has exhibited similar or even superior performance compared to human capabilities over a broad spectrum of workloads. However, for applications like smart devices, wearables for healthcare monitoring, or autonomous drones for spatial exploration that require constant real-time information processing, we want to embed implementation of neural networks on the edge. This imposes stringent constraints in terms of power, latency, and footprint area and requires us to rethink the approach toward building hardware for deep learning. Although the architecture of deep neural networks like convolutional neural networks (CNNs) is strongly inspired by the cortical hierarchies, the implementation deviates significantly from the biological counterpart. One obvious point of difference is that neurons are implemented using continuous non-linear functions like sigmoid or ReLu, whereas biological neurons compute using asynchronous spikes that indicate the occurrence of an event. Using such asynchronous event-based information processing may significantly bring down the hardware resources in terms of computational power and footprint area. A recent work established a gain of 54% in area and 45% in power for 65 nm CMOS ASIC implementation of SNN over multi-layer perceptron (MLP) at iso-accuracy and similar architecture (Khacef et al., 2018). Furthermore, with event-based sensors like visual sensors having reached a matured state (Lichtsteiner et al., 2008), SNNs provide a natural choice to be interfaced with them. In the last decade, there has been enormous efforts to build and scale up neuromorphic hardware using CMOS based mixed-signal (Benjamin et al., 2014; Chicca et al., 2014; Park et al., 2014; Qiao et al., 2015) and fully digital (Merolla et al., 2014; Davies et al., 2018) designs. However, there lies several considerations for hardware implementation of SNN that must be undertaken to minimize hardware resources (area and energy), some of which are discussed below.

One major consideration is the choice of the neuronal model and its hardware emulation either in analog or digital domain that will ultimately dictate the compactness and energy efficiency. Biological neurons consist of thin lipid layer membrane whose potential is altered by the arrival of excitatory or inhibitory post-synaptic potentials (PSPs) through the dendrites of the neuron. Upon sufficient stimulation, the neuron generates an action potential and the event is commonly referred to as *firing* or *spiking* of the neuron. To emulate these neuronal dynamics in a hardware, including the transient dynamics as well as the mechanism for neurotransmission, the first ingredient of the implementation is an appropriate choice of the neuron model. Although numerous models have been proposed by drawing inspiration from neuroscience like the biologically plausible complex Hodgkin–Huxley model (Hodgkin and Huxley, 1952) and the Izhikevich model (Izhikevich, 2003), we choose the bio-inspired leaky-integrate-and-fire (LIF)

neuron model that provides reduced complexity for hardware implementation while producing the required key dynamics for computation. Spiking LIF neuron can be implemented either in analog or digital domain. While fully digital spiking neurons have been implemented (Merolla et al., 2014; Davies et al., 2018), using analog circuits provides an alternative promising pathway. By using transistors biased in the sub-threshold regime, exponential behaviors can be easily mimicked allowing non-discretized continuous-time neural emulation (Indiveri, 2003; Chicca et al., 2014; Park et al., 2014; Qiao et al., 2015). Recently, Joubert et al. (2012) provided a quantitative comparison between a digital and analog implementation of LIF neuron at 65 nm CMOS technology node with the same level of performance and established an area and energy benefit of 5x and 20x, respectively, for analog over digital design. One pitfall for analog implementation is, however, the usage of large capacitors for emulating the membrane potential. Even with the most drastically scaled technology node, realizing dense on-chip capacitance comparable to biological neuronal membranes ($\sim 10 \text{ fF}/\mu\text{m}^2$; Gentet et al., 2000) is challenging. For example, Joubert et al. (2012) implemented the temporal integration property of an analog spiking neuron using a 500 fF metal-insulator-metal (MIM) capacitor that requires $100 \mu\text{m}^2$ silicon area while Indiveri et al. (2006) reports using a 432 fF capacitance occupying $244 \mu\text{m}^2$ silicon area. Additionally, biological neurons have been shown to be stochastic and this stochasticity adds to the richness of biological computation. With the recent focus on exploiting the physics of functional materials such as ferroelectrics, magnetics, and phase-change materials to build nano-scale devices that can emulate the characteristics of a low-power, stochastic, and capacitor-less spiking neuron, several proposals have been put forward (Sengupta et al., 2016; Tuma et al., 2016; Jerry et al., 2017). In this work, we experimentally demonstrate the essential neuronal dynamics in a 28 nm ferroelectric field-effect transistor (FeFET) technology with ultra-scaled gate length. The membrane potential is represented using the intrinsic ferroelectric polarization and the rich polarization switching dynamics is utilized to perform temporal integration of post-synaptic spikes, thus mimicking an LIF neuron.

The second consideration is the design of synaptic weight storage. Conventional von-Neumann architecture suffers from time and energy spent in moving data between a centralized memory and the processing units. In contrast, a non-von-Neumann architecture allows computation to be done at the location of the stored synaptic weights, thus circumventing the problem of data-movement. Typical examples of such neuromorphic hardware implementing distributed computing include Intel's Loihi chip with 128 cores each having a local 2 MB static random access memory (SRAM) (Davies et al., 2018) and IBM's TrueNorth with 4096 neurosynaptic cores each containing 12.75 kB local SRAM (Merolla et al., 2014; Akopyan et al., 2015). Additionally, novel techniques such as time-multiplexing has been proposed to reduce hardware resources or facilitate memory usage efficiently (Akopyan et al., 2015; Davies et al., 2018; Wang et al., 2018; Abderrahmane et al., 2020). Further improvement in energy efficient on-chip training and inference can come from replacing digital SRAM arrays with high density

analog synapses that can encode the synaptic weight directly using a physical property of the device such as conductance. Such analog synaptic weight cells can substantially reduce power for both training and inference (Morie and Amemiya, 1994; Burr et al., 2015; Gokmen and Vlasov, 2016). Desirable characteristics of such analog devices include fast and low-power programming of multiple analog states (bit resolution), good retention of the multiple states, and high endurance. Specifically for achieving on-chip training, gradual and symmetric conductance update characteristic is extremely crucial. Recent research efforts have explored numerous potential candidates for building such analog synaptic weight cells including resistive random access memory (RRAM) (Yu et al., 2015; Gao et al., 2015; Prezioso et al., 2015; Wu et al., 2017), phase-change memory (PCM) (Kuzum et al., 2012; Burr et al., 2015; Ambrogio et al., 2018) and FeFETs (Jerry et al., 2018a, 2019; Luo et al., 2019; Sun et al., 2019). In this work, we provide new experimental results of a FeFET-based synaptic weight cell at scaled device dimensions using 28 nm FeFET technology.

Finally, while deep learning, involving non-spiking and often CNNs, has made remarkable progress in achieving human-like performance at solving complex tasks, similar efficient training algorithms have been challenging to design for SNNs. The difficulty in applying traditional deep learning algorithms stems from various factors. First, the notion of time is an important aspect of SNN. As such, a different cost function has to be used that incorporates the notion of time while learning spatiotemporal patterns rather than what's commonly used in deep learning. Second, spiking neurons are inherently non-differentiable during their time of spike. Over the recent years, several efforts on training SNNs have been undertaken. These include indirect supervised learning like DNN to SNN conversion (O'Connor et al., 2013; Pérez-Carrasco et al., 2013; Diehl et al., 2015; Sengupta et al., 2019), direct supervised learning such as spatiotemporal backpropagation (Wu Y. et al., 2018; Wu J. et al., 2019), and unsupervised training of SNNs using bio-inspired local Hebbian learning rule like spike-time-dependent-plasticity (STDP) (Diehl and Cook, 2015; Panda and Roy, 2016; Kheradpisheh et al., 2018). In this work, we focus on the direct supervised learning scheme. Recently, Zenke and Ganguli (2018) proposed a novel supervised learning algorithm to train multilayer SNNs using a surrogate gradient (SG) based on the membrane potential, known as SuperSpike. In this work, we follow their approach closely by substituting the non-differential derivative of the step-function in the backward pass with a normalized negative part of a fast sigmoid of the membrane potential. Furthermore, we account for the limited bit precision offered by the FeFET-based synapses by considering weight quantization during the training process itself (Wu S. et al., 2018). We quantize the weights in the forward pass while working with high precision gradients. Previous works have shown that DNNs are very well capable of achieving state-of-the-art results with limited precision weights and activations (Choi et al., 2019) as well as quantized errors and gradients (Wu S. et al., 2018). Most modern quantization schemes require additional modification of the learning and inference process by scaling, clipping, or stochastic rounding of variables. Choi et al. (2019),

for example, train a separate parameter exclusively for activation clipping and compute a scaling factor for the weights that minimize quantization error. However, note that Choi et al. achieve good results with weights quantized using 2 bits in the forward and backward pass by storing and updating a full precision copy of the weights as well as quantizing them under the consideration of the first and second moment of the weight distribution (SAWB). Since spikes only require 1 bit, the energy consumption in our case is mainly driven by weights. Hence, we focus exclusively on the weight quantization and use the weight quantization method as described by Wu S. et al. (2018) since it imposes marginal quantization overhead. We perform supervised learning on MNIST dataset as an example. We further discuss the impact of FeFET device scaling on the achievable number of analog synaptic weight states (bit resolution) leading to a loss of classification accuracy and potential new avenues for research to circumvent the problem.

MATERIALS AND METHODS

FeFET-Based Analog Adaptive Spiking Neuron

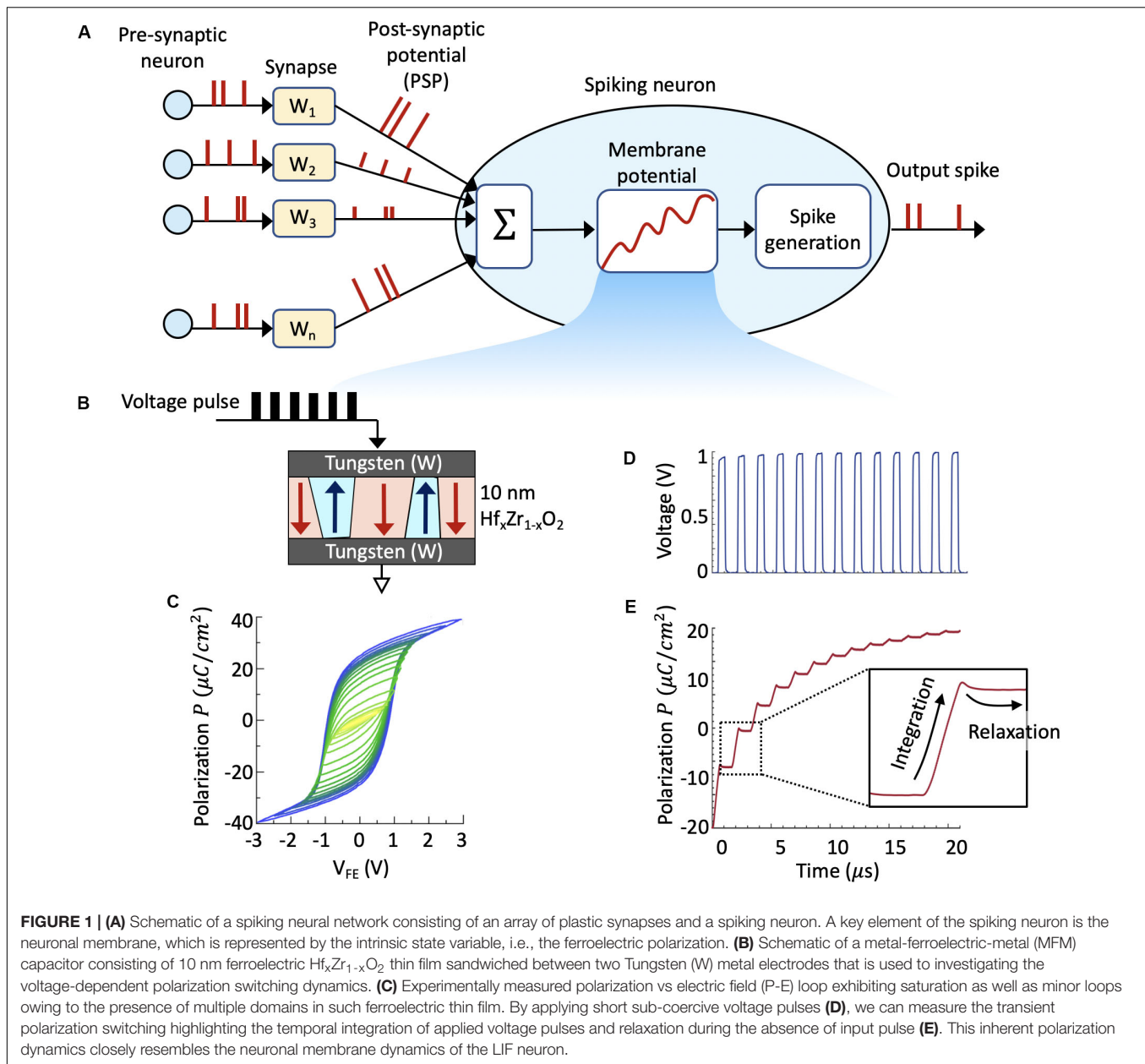
The general working principle of a SNN is as follows. When a synapse receives an action potential, also known as a spike, from its pre-synaptic neuron, it emits a PSP. The PSP in turn stimulates the membrane potential of the post-synaptic neuron. The neuronal membrane potential exhibits temporal evolution where it integrates the PSPs. When the membrane potential crosses a threshold, the post-synaptic neuron fires, i.e., it emits an output spike. **Figure 1A** illustrates the operation of a simple LIF neuron. Considering a generic LIF neuron, the membrane potential u is governed by the following equation:

$$\tau \frac{du}{dt} = f(u) + \sum w_i I_i$$

where $f(u)$ is the leak term accounting for the leakage of accumulated charge in the cell membrane, w_i is the synaptic weight, and I_i is the input current that depends on the excitatory or inhibitory PSPs. Upon arrival of excitatory input voltage pulses, the membrane potential continuously evolves in time and as it crosses a threshold, the neuronal circuit sends out an output voltage pulse thereby creating a “firing event.” The key idea behind a FeFET-based spiking neuron is to represent the membrane potential u by the intrinsic state variable, i.e., ferroelectric polarization instead of the charge stored by a capacitor. As will be discussed next, such dynamics can be achieved within the ferroelectric gate stack of a FeFET that allows realizing compact and low-power spiking neuron.

Polarization Switching Dynamics

We start by investigating the voltage-dependent polarization switching dynamics in a 10nm ferroelectric $\text{Hf}_x\text{Zr}_{1-x}\text{O}_2$ thin film sandwiched between two Tungsten (W) metal electrodes. Such a metal-ferroelectric-metal (MFM) capacitor is illustrated in **Figure 1B**. The fabricated capacitors have lateral dimensions of $80 \mu\text{m} \times 80 \mu\text{m}$. **Figure 1C** shows the experimentally measured



polarization vs electric field (P-E) loop exhibiting saturation as well as minor loops owing to the presence of multiple domains in such ferroelectric thin film. Starting from a negative polarization state, where all the dipoles are pointing down, we apply as short voltage pulse. Since the coercive field (V_C) exhibits a Gaussian distribution in such multidomain thin film, the applied short voltage pulse becomes larger than V_C in some of the domains leading to a partial polarization switching in the MFM capacitor.

In order to study the temporal evolution of polarization switching, next we apply short sub-coercive voltage pulses (**Figure 1D**) and measure the net switching current I_{Total} as a function of time. The total measured current I_{Total} will have contribution from two factors—the ferroelectric switching and the linear dielectric response. We subtract the contribution

from the dielectric portion to reveal the switching dynamics associated with the polarization alone. **Figure 1E** shows the transient polarization switching dynamics highlighting the temporal integration of applied voltage pulses and relaxation during the absence of input pulse. The neuronal dynamics is emulated by utilizing the ferroelectric polarization accumulation property (Ni et al., 2018; Saha et al., 2019) that allows temporal integration of PSP. Such ferroelectric polarization switching dynamics bear close resemblance to that of a LIF spiking neuron. It is intriguing to compare the dynamics of the FeFET-based neuron with a standard LIF neuron realized using dielectric capacitor. It is seen that the integration behavior is similar for both the neurons. However, the leak characteristics indicate a surprisingly opposite behavior. As seen in **Figure 1E**,

the transient relaxation in ferroelectric polarization when the voltage pulse is switched off decreases with the increasing in the number of applied pulses. This is contrary to that of a standard LIF neuron built using linear dielectric capacitor, where the discharge rate of the capacitor increases with the pulse number. Such a deviation in polarization relaxation dynamics can be understood by considering the interaction among the ferroelectric domains within the thin film and has been recently studied using phase-field modeling approach (Dutta et al., 2019b; Saha et al., 2019).

FeFET Switching Dynamics

Next, we extend the investigation of polarization switching dynamics to FeFETs that consist of a doped-HfO₂ ferroelectric layer integrated into the gate stack of a conventional MOSFET. **Figure 2A** shows the schematic and TEM of a high-K metal gate FeFET with a poly-Si/TiN/Si:HfO₂/SiON/p-Si gate stack fabricated at 28 nm technology node (Trentzsch et al., 2017). All experiments reported here have been performed on FeFETs with channel length of 34 nm and width of 80 nm. On application of successive sub-coercive voltage pulses to the gate of FeFET, the ferroelectric polarization within the Si:HfO₂ layer switches due to an accumulative effect (Mulaosmanovic et al., 2018a; Ni et al., 2018; Saha et al., 2019), resulting in the modulation of the threshold voltage (V_T) of the FeFET. As seen in **Figure 2B**, the V_T gets modulated abruptly from a high- V_T to low- V_T state. The abrupt V_T shift arises due to the presence of very few grains (hence ferroelectric domains) within such a scaled device. This in turn causes an abrupt increase in the drain-to-source channel conductance (G_{DS}), thereby exhibiting the temporal integration of PSPs in FeFET. **Figure 2C** shows the measured conductance modulation as a function of the number of applied pulses over multiple cycles. The cycle-to-cycle variation arises from the nucleation dominated ferroelectric polarization switching in FeFET which at the domain level is known to be a stochastic process (Mulaosmanovic et al., 2018b; Dutta et al., 2019a; Ni et al., 2019a). Once G_{DS} exceeds a threshold, the drain current (I_D) increases and the FeFET is said to “fire.” Once in the low- V_T state, a negative voltage needs to be applied across the gate and drain/source in order to reset the FeFET to high- V_T state. Additionally, since the V_T can be gradually increased as well as decreased by applying positive and negative voltage pulses, respectively, this allows the incorporation of both excitatory ($I > 0$) and inhibitory ($I < 0$) inputs without any additional circuitry. **Figure 2D** shows the continuous conductance modulation due to the application of PSPs and how the integrate-and-fire (IF) dynamics repeats after each reset. Owing to the inherent stochasticity, over multiple IF cycles, a single neuron exhibits a distribution of inter-spike intervals for a range of applied input voltage pulse amplitude or width. **Figures 2E,F** show the distribution of inter-spike interval for a range of voltage amplitudes and the corresponding stochastic firing rate of the neuron. Similar impact of varying the input pulse width on the inter-spike interval and firing rate is shown in **Figures 2G,H**. Such stochasticity can be harnessed for emulating the probabilistic activity exhibited by biological neurons (Faisal et al., 2008)

without implementing any additional complex circuitry for randomness generation.

Implementation of Adaptive Spiking Neuron

We leverage this rich dynamics of the FeFET to implement a low-power spiking neuron circuit consisting of three transistors and one FeFET. Utilizing the temporal integration property of FeFET also allows us to avoid using capacitors for membrane potential, thus providing us an area advantage as well. **Figure 3A** illustrates the proposed neuron circuit. The input PSPs are applied to the PMOS M1. Initially, both the node voltages V_0 and V_1 are at 0 V. As the PSPs are applied, the node voltage V_0 increases and sub-coercive voltage pulses are applied to the gate of FeFET. Upon application of successive pulses, the FeFET abruptly changes from high- V_T to low- V_T state and the drain current I_D increases. This sends out an output voltage pulse (“spike”) as well as increases the node voltage V_1 . Once an output spike is generated, a reset signal is applied to transistor M3. This external reset, initiated by an arbiter, enables array-based operation often seen in large-scale, event-driven, asynchronous systems such as (Indiveri et al., 2011; Benjamin et al., 2014; Park et al., 2014). With M1 being cut-off during the inter-spike intervals, the node voltage V_0 is pulled down to 0 V. This results in a negative V_{GS} across the FeFET, thus switching the polarization in opposite direction and resetting the FeFET to high- V_T state. We also incorporate bio-inspired homeostatic mechanism that regulates the activity of a neuron and lowers the firing rate after every output spike (Liu and Wang, 2001; Benda and Herz, 2003). The homeostatic spike frequency adaptation mechanism is introduced through three additional transistors M4–M6 as shown in **Figure 3A**. The capacitance C_P can be realized by considering the parasitic capacitance of that node. During every output spike event, as the node voltage V_1 goes high, transistor M4 gets turned on and that in turn increases the node voltage V_2 . The discharge rate of V_2 can be controlled by adding an additional transistor. As V_2 increases, M5 gets turned on gradually with every output spike which in turn increases the discharge rate of node voltage V_0 . Thus, the neuron has to integrate over more input PSPs in order to spike which brings down the neuron’s firing rate with every output spiking event, thereby implementing spike frequency adaptation. **Figure 3B** shows the SPICE circuit simulation of the FeFET-based adaptive spiking neuron. To mimic the stochastic switching dynamics of the FeFET, we introduce a distribution of the coercive field (V_C) for the ferroelectric domains. We performed Monte Carlo simulation to generate the stochastic spike frequency adaptation as shown in **Figure 3C**, where the instantaneous firing rate goes down with each output spike. The implication of such a stochastic neuron on the classification accuracy is discussed in section “Results.”

FeFET-Based Analog Synapse

The idea of voltage-dependent partial polarization switching in ferroelectric Hf_xZr_{1-x}O₂ can be leveraged to implement a non-volatile FeFET-based analog synapse. As illustrated in **Figure 4A**, the FeFET synapse can be integrated into a pseudo-crossbar

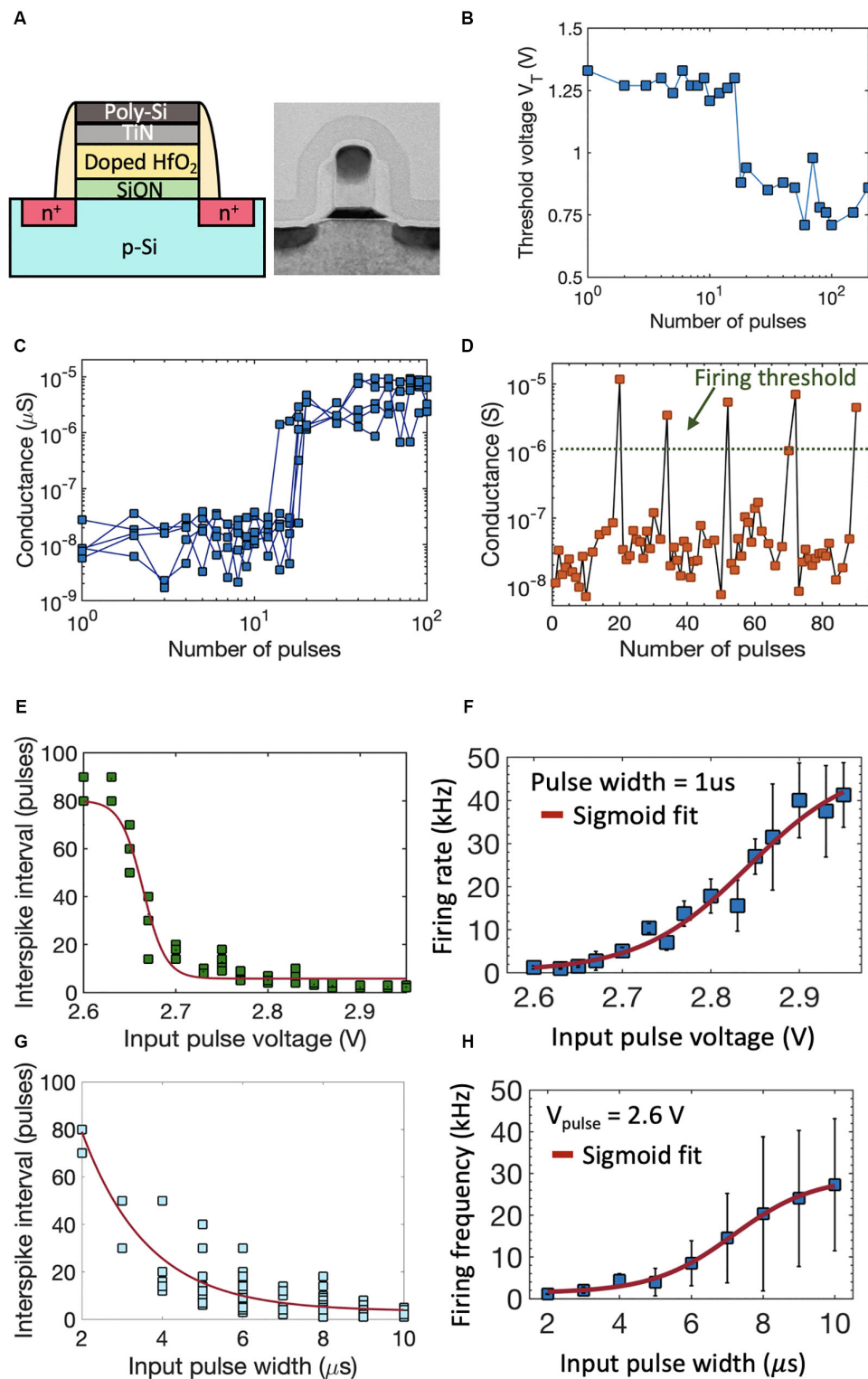
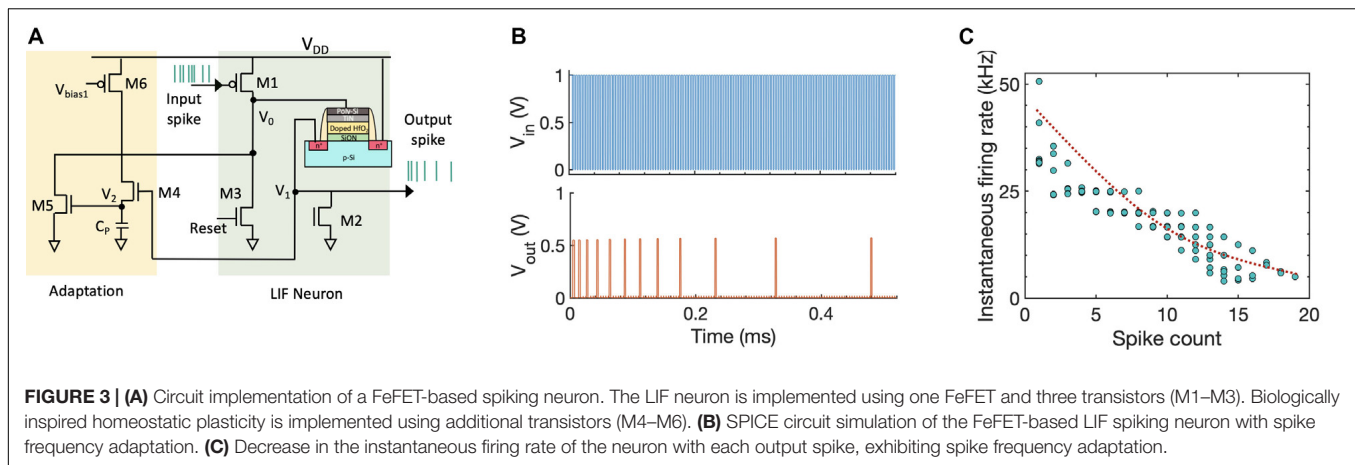


FIGURE 2 | (A) Schematic and TEM of a high-K metal gate FeFET with a poly-Si/TiN/Si:HfO₂/SiON/p-Si gate stack fabricated at 28 nm technology node. **(B)** On application of successive sub-coercive voltage pulses to the gate of FeFET, the threshold voltage V_T gets modulated abruptly from a high- V_T to low- V_T state. **(C)** Corresponding conductance modulation as a function of number of applied pulses, measured over multiple cycles. **(D)** The integrate-and-fire dynamics of the FeFET neuron. After reaching a conductance threshold, the FeFET is reset to the initial polarization state using a negative gate voltage, which results in a sequence of firing events. **(E,F)** Distribution of inter-spike interval for a range of voltage amplitudes and the corresponding stochastic firing rate of the FeFET neuron. Similar impact of varying the input pulse width on the inter-spike interval and firing rate is seen in **(G,H)**.



array that is suitable for row-wise weight update and column-wise summation. Recently, FeFET-based analog synapse has been experimentally demonstrated on 3 μm long and 20 μm wide devices that exhibited 32 non-volatile states (equivalent to 5-bit precision) and a dynamic range of 45x with amplitude modulated programming pulses (Jerry et al., 2018a,b). Here, we provide experimentally measured conductance modulation in a scaled 500 nm \times 500 nm high-K metal gate FeFET fabricated at 28 nm technology node (Trentzsch et al., 2017). As shown in **Figure 4B**, we used the amplitude modulation scheme with pulse widths of 1 μs to modulate the conductance of the FeFET. Applying progressively increasing gate pulses V_P causes the FeFET to transition from the initial high- V_T state to lower V_T states as shown by the I_D - V_G characteristics in **Figure 4B**. The resulting channel conductance G_{DS} progressively increases as shown in **Figure 4C**. However, due to the lateral scaling of the device, the number of ferroelectric domains decreases resulting in a reduced number of non-volatile states. Since the typical grain size in 10 nm HfO_2 is around 10–15 nm, it can be estimated that there will be around 1000 domains for a 500 nm \times 500 nm FeFET. This also results in cycle-to-cycle (as well as device-to-device) variation, since the stochastic domain switching contribution from individual domains becomes more pronounced (Ni et al., 2019a). The inherent stochasticity results in a variation of the conductance states measured over multiple cycles for each voltage applied as shown in **Figure 4C**. We choose eight non-overlapping G_{DS} states obtained over multiple cycles using both potentiation and depression pulses as shown in **Figure 4D** that allowed the representation of a 3-bit equivalent analog weight cell. **Figure 4E** shows the cumulative distribution of the G_{DS} states corresponding to potentiation pulse scheme obtained over multiple cycles. This indicates that while FeFETs are a promising candidate for non-volatile analog synapse, the number of available non-volatile states drastically reduce at the scaled node (Dutta et al., 2019a; Ni et al., 2019a). The implications of such a reduced bit-precision on learning algorithms are discussed next. This challenge also opens up new avenues of research both at the material/device/circuit level, as well as at the algorithmic level. For example, a FeFET-based synapse has

been recently proposed that utilizes hybrid precision training and inference to overcome the challenge of limited bit precision (Sun et al., 2019).

Model of FeFET-Based Analog Spiking Neuron and Synapse

The inherent ferroelectric polarization switching dynamics closely resembles the neuronal membrane dynamics of the LIF neuron and can be captured by a modified quasi-LIF neuron model. Our description of the FeFET-based neuron model builds upon the traditional LIF neuron presented by Diehl and Cook (2015) as given below:

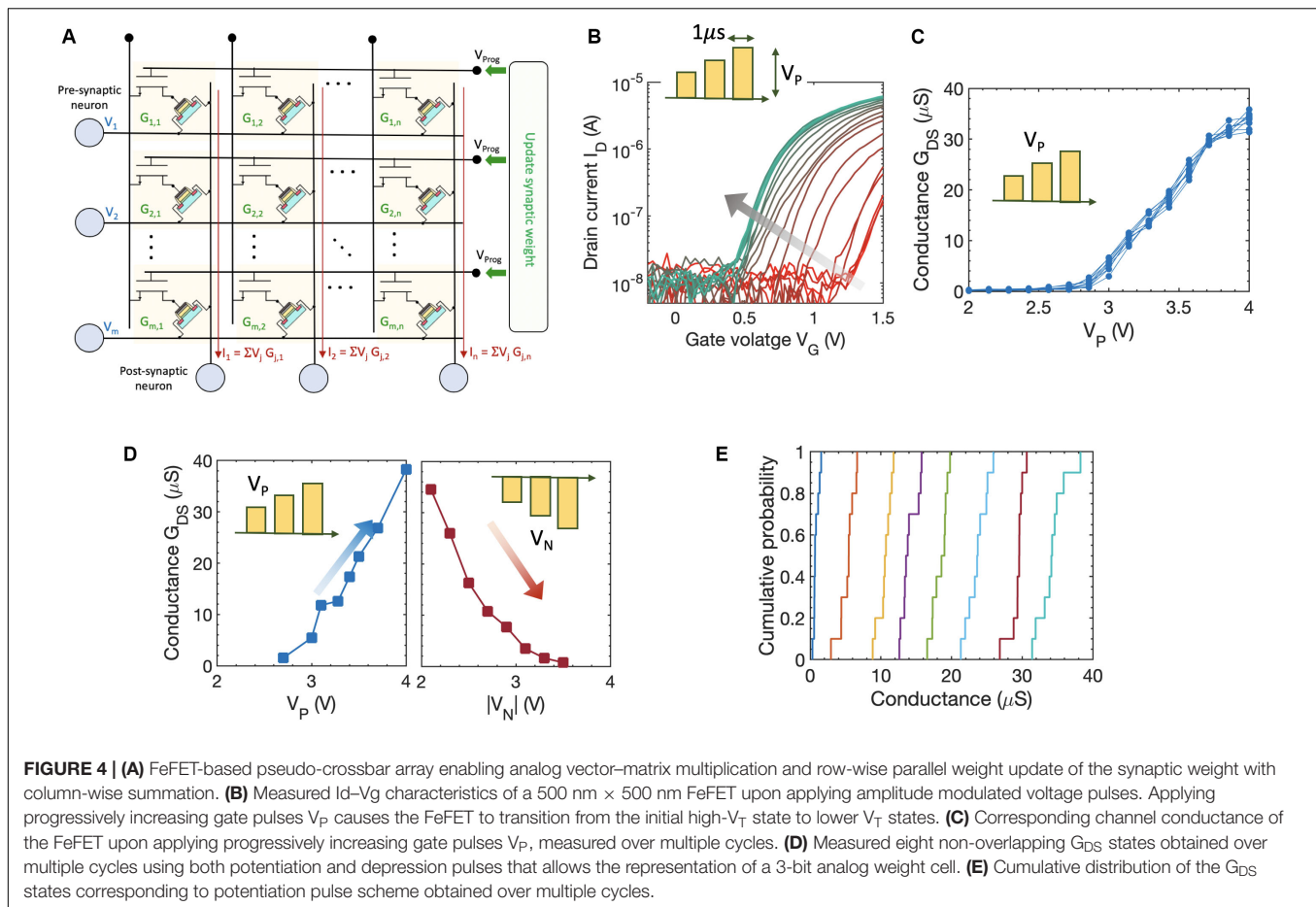
$$\frac{dv}{dt} = \frac{\alpha v_{rest} - v}{\tau_{leak}} + \frac{\sum (g_e (E_{exc} - v) + g_i (E_{inh} - v))}{\tau_{integrate}}$$

$$\tau_{ge} \frac{dg_e}{dt} = -g_e$$

$$\tau_{gi} \frac{dg_i}{dt} = -g_i$$

$$\tau_{\alpha} \frac{d\alpha}{dt} = -g_e$$

where v is the membrane potential, v_{rest} denotes the resting potential of the neuron, and E_{exc} and E_{inh} are the equilibrium potentials of excitatory and inhibitory synapses. τ_{leak} and $\tau_{integrate}$ are the time constants associated with for the leakage and integration phase of the neuron, respectively. When the neuron's membrane potential crosses the membrane threshold v_{thres} , the neuron fires and the membrane potential is reset to v_{reset} . We incorporate the quasi-leaky behavior (decrease in the leak rate of the neuron during the inter-spike interval) into the neuron model by using a variable resetting voltage by multiplying v_{reset} with a parameter α that changes with each incoming spike. Additionally, this quasi-leak behavior can also be incorporated into the model by using a variable τ_{leak} that also depends on the membrane potential v (Dutta et al., 2019b). However, owing to very small relaxation dynamics, one can also ignore the leaky behavior of



the FeFET-based neuron and treat it as a perfect IF neuron (Mulaosmanovic et al., 2018a). Furthermore, we use an adaptive threshold regime to regulate the neuron's activity. Once a neuron hits the threshold and issues a spike, this neuron's threshold increases by a fixed amount, thereby making it harder for this neuron to spike again and prioritizing activities of other neurons. However, the threshold increase only happens until the neuron threshold reaches a maximum level, after which the threshold is not changed by issued spikes anymore.

Synapse models have been incorporated following Diehl and Cook (2015) where the synaptic conductance changes instantaneously by weight w when a presynaptic spike arrives at the synapse, else the conductance decays exponentially. g_e and g_i are the conductances of the excitatory and inhibitory synapse, respectively. τ_{ge} and τ_{gi} are the time constants of the excitatory and inhibitory PSPs, respectively.

This model is then discretized with a standard Euler method so we can use discrete time steps in our simulation. The discrete time version of the models is expressed as:

$$\begin{aligned} v[n+1] &= \alpha E_{rest} + \beta v[n] + g_e[n] E_{exc} \\ &\quad + \beta g_e[n] v[n] + g_i[n] E_{inh} + \beta g_i[n] v[n] \\ g_e[n+1] &= e^{-\frac{\Delta t}{\tau_{ge}}} g_e[n] \end{aligned}$$

$$g_i[n+1] = e^{-\frac{\Delta t}{\tau_{gi}}} g_i[n]$$

$$\alpha[n+1] = e^{-\frac{\Delta t}{\tau_\alpha}} \alpha[n]$$

where $v[n]$ is the discretized membrane potential of the neuron at time step n . We use a single membrane time constant τ_v , accounting for both τ_{leak} and $\tau_{integrate}$. $\beta = e^{-\frac{\Delta t}{\tau_v}}$ captures the decay in the membrane potential during a Δt time step.

Supervised Learning for SNNs

The success of deep learning in recent years has largely been attributed to the power of supervised learning techniques and gradient based learning (Lecun et al., 2015). Given an objective function, backpropagation adjusts parameters and weights of a given network so that its objective function is minimized. In DNNs, weights are updated in multiple layers organized hierarchically enabling it to learn complex classification or regression functions. Two critical challenges must be overcome in order for SNNs to gain similar success: (a) The development of hierarchical "deep" networks like (Panda and Roy, 2016; Kheradpisheh et al., 2018) which can learn complex representations and (b) enabling the application of

gradient based learning to deep spiking neural networks (SSNs). Several studies have proposed ways to train SSNs in order to address the above challenges, such as (Gütig, 2014; Anwani and Rajendran, 2015). However, these approaches did not sufficiently enhance the representative power of SSNs, nor did they enable the development of deeper more complex networks. Neftci et al. (2019) identified four streams of research attempting to train SSNs with hidden units: (i) biologically inspired local learning rules, (ii) translating conventionally trained “rate-based” neural networks to SSNs, (iii) smoothing the network model to be continuously differentiable, and iv) defining a SG as a continuous relaxation. Fortuitously, the SG-based methods simultaneously address the two challenges presented and form the basis for the remainder of this article. This method allows us to build upon the solid research base of backpropagation with only marginal modifications of the spiking network model.

Surrogate Gradient Learning

Historically, the spike function in SSNs prevented the application of gradient based learning rules due to the discontinuities induced by the non-differentiable spikes, consequently “stopping the gradient from flowing.” This in turn results in backpropagation failing to function correctly. The SG method substitutes the gradient in the backward pass of the backpropagation with a differentiable proxy or surrogate. This surrogate is generally based on the membrane potential of a neuron. As a result of this new gradient, the non-differentiability is circumvented, and the gradient can propagate. Using this gradient-based update rule, standard solutions to the credit assignment problem can be applied. Thus, given a global loss function, we can apply traditional gradient-based learning methods such as backpropagation through time (BPTT) (Huh and Sejnowski, 2018) or other learning rules such as three factor learning rules (Zenke and Ganguli, 2018) to SSNs. Most modern machine learning libraries (e.g., PyTorch or TensorFlow) provide autograd functionalities which facilitate the gradient computation. Our experiments used BPTT as gradient-based learning method in conjunction with a SG in the backward pass. The SGs were computed by applying the normalized negative part of a fast sigmoid on the membrane potential.

Quantization

As highlighted earlier, the on-chip FeFET-based analog synapse provides limited bit precision ranging from 5 to 3 bits in scaled devices. Efficient implementation of (i) off-chip training followed by reduced bit precision for inference mode and (ii) on-chip learning and inference with reduced bit precision, both demand efficient training algorithm taking into consideration the quantization in synaptic weights. To accurately model the effect of quantizing the weights, we follow the procedure outlined in Wu S. et al. (2018). Weights are quantized by restricting them to a feasible range $[-1 + \sigma(b_w), +1 - \sigma(b_w)]$, where $\sigma(b) = 2^{1-b}$ and b_w is the number of bits encoding the weight. Weights in each layer are further scaled by γ :

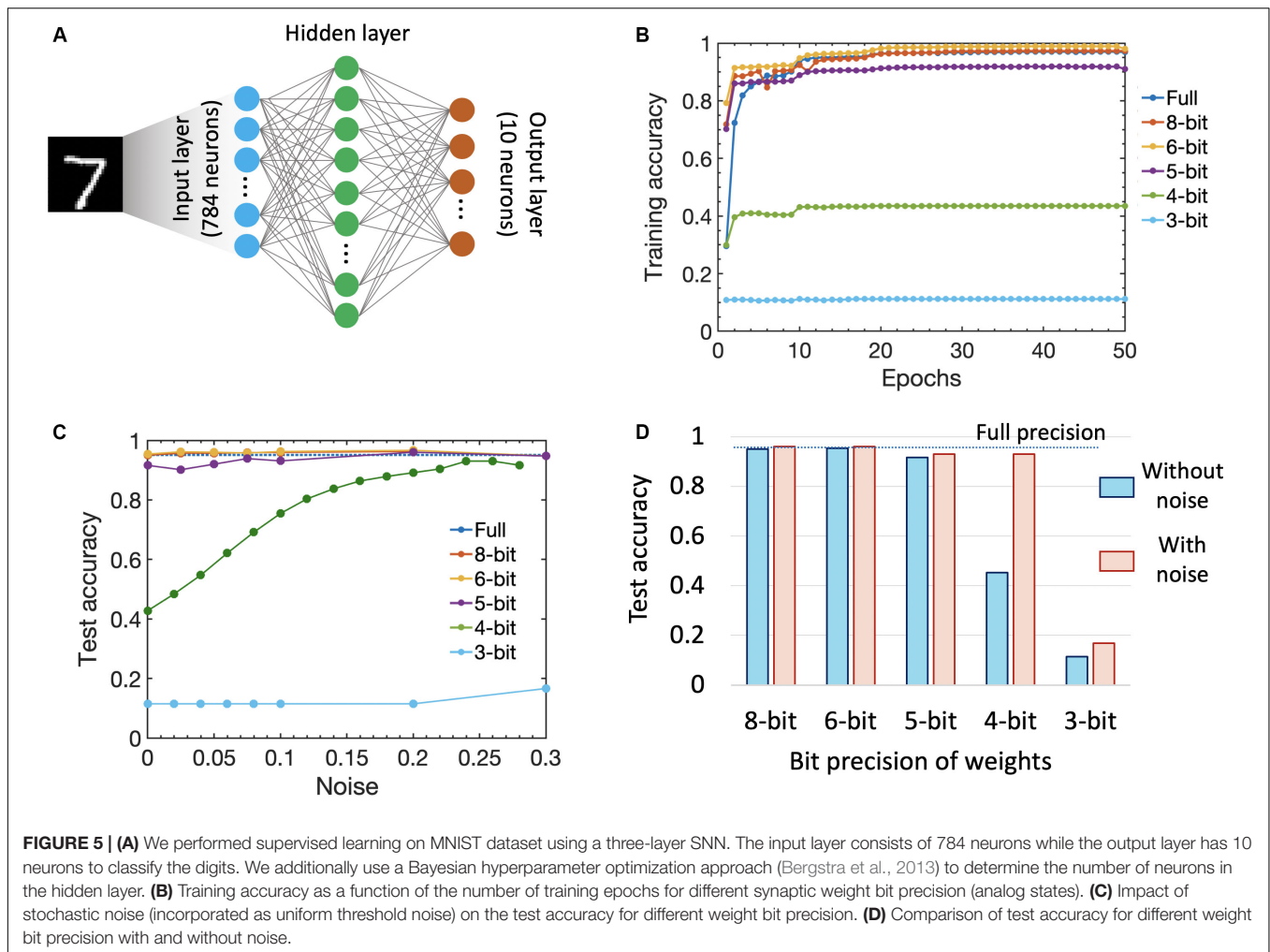
$$\gamma = 2^{\text{round}\left[\log_2\left(\frac{\left(\frac{1}{\sigma(b_w)} - 0.5\right)\sigma(b_w)}{\sqrt{\frac{3}{fan\ in}}}\right)\right]}$$

where *fan in* represents the number of connections into a layer. Weights are also uniformly initialized in their feasible range and clipped to the range after each update during training.

RESULTS

We performed supervised learning on MNIST dataset (using the standard train/test split of 60,000/10,000) using a three-layer SNN as shown in **Figure 5A**. The input layer consists of 784 neurons while the output layer has 10 neurons to classify the digits. The number of hidden layer units is an architectural question which can have significant impact on the performance. We used a Bayesian hyperparameter optimization approach (Bergstra et al., 2013) to determine the number of hidden layer neurons, learning rate, input multiplier (scaling of the input spikes), scaling coefficient for the τ_{leak} in relation to $\tau_{integrate}$, size of the regularizer, batch size, and saturation threshold. All these parameters have an impact on the performance and are sensitive to the dataset of the network. We use the saturation threshold method taken from Yousefzadeh et al. (2018) which prevents the firing threshold from being further increased once it surpasses the saturation threshold, e.g., once it issued a certain number of spikes. For the hyperparameter optimization, we gave the optimizer ranges for the mentioned parameters and programmed it to train a sampled configuration for 12 epochs. Overall, we constrained the optimizer to use 75 evaluations and come up with the best configuration. The number of discrete time steps remained fixed at 80. In our simulations, we used a negative loglikelihood loss function, which performed classification by integrating the last layer’s neurons membrane potential over time and selecting the class of the neuron with the largest integration value.

Since the number of analog states that can be represented by a single FeFET-based synaptic weight cell decreases as we scale the device, it is important to consider the impact of bit precision of the synaptic weights (number of analog states) on training as well as test accuracy. Hence, in our simulation, we varied the weight bit precision from a high value of 8 bits (that would require a single FeFET to represent 256 analog states) to 5 bits (32 analog states demonstrated in a $3\ \mu\text{m} \times 20\ \mu\text{m}$ device (Jerry et al., 2018a,b) down to 3 bits (eight analog states demonstrated in this work). We also compared the results against the full 32-bit floating point precision available on a CPU. **Figure 5B** shows the training accuracy as a function of the number of training epochs for various precision of the synaptic weight without introducing any stochastic noise in the simulation. It is seen that while up to 6 bits, we get a test accuracy of 95.4% which is comparable (or even better) to that of the full precision accuracy of 95.1%. The accuracy starts decreasing to 91% for 5 bits and drastically down to 43.5% for 4 bit precision. The increase of accuracy at 6 bits can be seen as the regularizing effect of more coarse weights. The sharp decrease of accuracy for 5 or fewer bits likely indicate the tolerance threshold of SSNs toward reduced weight precision and the lack of information in spikes coupled with weights after a certain weight quantization level. Note that previous works like Choi et al. achieve good results with 2-bit weight quantization



in the forward and backward pass by storing and updating a full precision copy of the weights as well as quantizing them under the consideration of the first and second moment of the weight distribution (SAWB). In contrast, our results are obtained with only quantized weights in both the forward and backward pass as well as a linear quantization step reflecting the capabilities of our proposed device.

We further studied the impact of stochastic neurons on the overall performance of the SNN by introducing a uniform noise around the membrane threshold which can be mimicked by the stochastic neuronal dynamics (as shown in Figure 2). Figure 5C shows the impact of noise on the test accuracy. The accuracy for 5–8 bits increased to 96%. Interestingly the accuracy for 4-bit weights improved substantially with more noise around the threshold which is in accordance with previous works on ordinary quantized DNNs (Wu S. et al., 2018; Choi et al., 2019). Over a population of neurons and multiple times steps, the threshold with more noise becomes more like a soft function (e.g., sigmoid, softmax, or tanh) rather than a hard threshold and hence more similar to ordinary DNNs which allows for reduced weight precision. In the case of 3-bit weights, we were not able to compensate for the granularity of the weights with noise around

the threshold. Figure 5D shows the testing accuracy as a function of various weight precision with and without noise indicating that having a stochastic SNN helps improve the classification accuracy in the presence of reduced weight precision. As mentioned earlier, another way to further improve the accuracy will be to resort to a CMOS-augmented FeFET-based hybrid synapse design that can provide hybrid precision training and inference to overcome the challenge of limited bit precision (Sun et al., 2019).

DISCUSSION

In this work, we exploit the rich dynamics of ferroelectric polarization switching in FeFET to realize compact and low-power analog spiking neuron and synapse. The membrane potential of the spiking neuron is represented by the intrinsic ferroelectric polarization of the FeFET. The neuronal dynamics is emulated by utilizing the polarization accumulation property (Ni et al., 2018; Saha et al., 2019) that allows temporal integration of PSP. This allows the realization of a capacitor-less analog spiking neuron which proves to be compact and low power. Table 1 shows a comparative study between our FeFET-based

TABLE 1 | Comparative study between various hardware implementations of spiking neuron.

	Indiveri et al., 2006	Joubert et al., 2012		Tuma et al., 2016	Sengupta et al., 2016	Jerry et al., 2017	This work
Neuron type	LIF	Analog LIF	Digital LIF	LIF	LIF	Piecewise linear FHN	LIF
Material	CMOS	CMOS	CMOS	Phase change (PCM)	Magnetic tunnel junction (MTJ)	Vanadium dioxide (VO ₂)	Ferroelectric HZO
Technology	800 nm	65 nm	65 nm	14 nm	–	–	45 nm
Integration mechanism	Capacitor charging	Capacitor charging	–	Joule heating	Magnetization dynamics	Capacitor charging	Polarization accumulative
Circuit elements	22 Transistor + one capacitor	33 Transistor + one capacitor	Pulse generator, counter, and comparator	One PCM + digital circuit	Two MTJs + four transistors	One VO ₂ + one transistor + one capacitor	One FeFET + six transistors
Stochasticity	Yes	No	No	Yes	Yes	Yes	Yes
Power or energy/spike	900 pJ	2 pJ	41.3 pJ	120 μ W	–	11.9 μ W	1–10 pJ
Firing rate	200 Hz	2 MHz	2 MHz	35–40 kHz	–	30 kHz	50 kHz
Area	2573 μ m ²	120 μ m ²	538 μ m ²	0.5–1 μ m ²	–	–	2.05 μ m ²

analog spiking neuron and various other proposals. Compared to CMOS-based realization of LIF neuron that requires more than 20 transistors and an explicit capacitor, our proposal of FeFET-based spiking neuron requires seven transistors including one FeFET. To estimate the areal requirements, we performed a layout using a 45nm technology node. The estimated area is approximately $1.74 \times 1.18 \mu\text{m}^2$ which would be much smaller than the capacitor-based CMOS circuits. For example, Joubert et al. (2012) realized an analog spiking neuron with a footprint area of $120 \mu\text{m}^2$ at 65 nm technology node, of which $100 \mu\text{m}^2$ was dedicated to realizing the 500 fF capacitor. Our estimated footprint area in terms of feature size F is at least 4x lower than this. Similarly, Indiveri et al. (2006) report using a 432 fF capacitance occupying $244 \mu\text{m}^2$ silicon area. The energy dissipated by our FeFET-based analog neuron is comparable to the analog neuron implementation by Joubert et al. (2012) while it is 4x lower than the digital implementation and 90x lower than the energy dissipated by Indiveri et al. (2006). Compared to PCM-based neuron that requires additional digital circuitry like a latch and a NOR logic gate (Tuma et al., 2016), our FeFET-based neuron dissipates 40x lower power and occupies at least 2.5x lower area in terms of feature size F. Compared to insulator-to-metal phase-transition vanadium dioxide (VO₂)-based neuron (Jerry et al., 2017), FeFET-based neuron dissipates 300x lower power.

TABLE 2 | Comparative study between various hardware implementations of analog synaptic weight cell.

	PCM	RRAM	FeFET
Material	GST	TaO _x /HfO _x	Hf _x Zr _{1-x} O ₂
States	8	128	8
Variation	~1.5%	~3.7%	<0.5%
Write voltage	2.5 V	1.6 V	4 V
Write energy	30 pJ	~10 pJ	0.1 pJ
Cell area	25F ²	24F ²	24F ²

The intrinsic ferroelectric polarization switching mechanism being a stochastic process (Mulaosmanovic et al., 2018b, Mulaosmanovic et al., 2018c; Dutta et al., 2019a; Ni et al., 2019a), the FeFET-based spiking neuron exhibits stochastic firing that maybe useful for building stochastic neural networks like neural sampling machine with novel properties like inherent weight normalization (Detorakis et al., 2019), for applications like modeling uncertainties in neural networks (Gal and Ghahramani, 2016) and for probabilistic inferencing (Pecevski et al., 2011). One key limitation of FeFET-based neuron compared to generalized neuron model utilized in neuroscience and CMOS-based circuits is that the membrane potential is represented by the intrinsic ferroelectric polarization state variable and the associated stochasticity arises directly from the ferroelectric domain nucleation process. Hence, the degree of tuning the neuronal parameters and the stochastic response is limited which might be disadvantageous for algorithms in which the parameters and the stochasticity have to be tightly controlled.

The FeFET-based analog synapse is realized using voltage-dependent partial polarization switching in multi-domain ferroelectric thin film (Jerry et al., 2018a,b). Recent experimental works have shown the ability to program FeFETs with voltage pulse widths as low as 50 ns (Jerry et al., 2018b) while the programming voltage can be brought down from 4 to 1.8 V by engineering the gate stack by adding an additional metal layer between the ferroelectric capacitor and MOS capacitor (Ni et al., 2019b). **Table 2** shows a comparative study between FeFET-based analog synapse and various other candidates like PCM (Burr et al., 2010; Athmanathan et al., 2016; Ambrogio et al., 2018) and RRAM (Lee et al., 2012; Wu et al., 2017; Wu W. et al., 2018; Luo et al., 2019). One major benefit of using FeFET for implementing analog synapse is the reduced variability to less than 0.5% (Luo et al., 2019) and an order of magnitude reduction in write energy (Dunkel et al., 2018; Ni et al., 2019b). The cell area is comparable to that of PCM and RRAM. One limitation of FeFET-based analog synapse is the achievable number of non-volatile conductance states as we scale down the

device. While a recent experiment on $60\ \mu\text{m}^2$ size FeFET devices exhibited 32 conductance states (equivalent to 5-bit precision) (Jerry et al., 2018a,b), in this work, we achieved eight non-overlapping conductance states (equivalent to 3 bits) in a $0.25\ \mu\text{m}^2$ size device. The precision of synaptic weight overed can be further improved by resorting to hybrid mechanisms like the recently proposed two transistor-one FEFET (2T1F) hybrid weight cell that allow up to 64 states with improved non-linearity and asymmetry factors (Luo et al., 2019; Sun et al., 2019). Similar hybrid schemes have been applied to other novel devices like the three-transistor, one-capacitor, and two PCM (3T1C+2PCM) weight cell (Ambrogio et al., 2018).

CONCLUSION

In summary, we explore the rich polarization switching dynamics and non-volatile nature of FeFETs and propose an all FeFET-based SNN neuromorphic hardware that can enable low-power spike-based information processing and co-localized memory and computing (a.k.a. in-memory computing). We experimentally demonstrate the essential neuronal and synaptic dynamics in a 28 nm high-K metal gate FeFET technology. Furthermore, we implement a SG learning algorithm on our SNN platform, thus enabling us to perform supervised learning. As such, the work provides a pathway toward building energy-efficient neuromorphic hardware that can support traditional machine learning algorithms. We also undertake synergistic device-algorithm co-design by accounting for the impacts of device-level variation (stochasticity) and limited bit precision of on-chip synaptic weights (available analog states) on the

classification accuracy and highlight possible avenues of future work to overcome the current challenges such as resorting to hybrid precision training and inference.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

SDu, SJ, and SDa developed the main idea. SDu performed all the measurements. SDu, JG, and KN performed the circuit simulations. CS performed the machine learning simulations. All authors discussed the results, agreed to the conclusions of the manuscript, and contributed to the writing of the manuscript.

FUNDING

This work was supported in part by the NSF sponsored ASSIST Engineering Research Center, Semiconductor Research Corporation (SRC), and DARPA.

ACKNOWLEDGMENTS

We are grateful to M. Trentzsch, S. Dunkel, S. Beyer, and W. Taylor at Globalfoundries Dresden, Germany, for providing 28 nm HKMG FeFET test devices.

REFERENCES

- Abderrahmane, N., Lemaire, E., and Miramond, B. (2020). Design space exploration of hardware spiking neurons for embedded artificial intelligence. *Neural Networks* 121, 366–386. doi: 10.1016/j.neunet.2019.09.024
- 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 neuromorphic chip. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396
- 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
- Anwani, N., and Rajendran, B. (2015). “NormAD - normalized approximate descent based supervised learning rule for spiking neurons,” in *Proceedings of the International Joint Conference on Neural Networks*, Killarney.
- Athmanathan, A., Stanisavljevic, M., Papandreou, N., Pozidis, H., and Eleftheriou, E. (2016). Multilevel-cell phase-change memory: a viable technology. *IEEE J. Emerg. Sel. Top. Circuits Syst.* 6, 87–100. doi: 10.1109/JETCAS.2016.2528598
- Benda, J., and Herz, A. V. M. (2003). A universal model for spike-frequency adaptation. *Neural Comput.* 15, 2523–2564. doi: 10.1162/089976603322385063
- 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
- Bergstra, J., Yamins, D., and Cox, D. D. (2013). “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures,” in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, Atlanta.
- Burr, G. W., Breitwisch, M. J., Franceschini, M., Garetto, D., Gopalakrishnan, K., Jackson, B., et al. (2010). Phase change memory technology. *J. Vac. Sci. Technol. B, Nanotechnol. Microelectron. Mater. Process. Meas. Phenom.* 28, 223–262. doi: 10.1116/1.3301579
- 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
- Chicca, E., Stefanini, F., Bartolozzi, C., and Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proc. IEEE* 102, 1367–1388. doi: 10.1109/JPROC.2014.2313954
- Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). “Accurate and efficient 2-Bit quantized neural networks,” in *SysML*, (Stanford, CA).
- 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
- Detorakis, G., Dutta, S., Khanna, A., Jerry, M., Datta, S., and Neftci, E. (2019). Inherent weight normalization in stochastic neural networks. *Adv. Neural Informat. Proc. Syst.* 3286–3297.
- 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 *Proceedings of the International Joint Conference on Neural Networks*, Killarney.

- Düinkel, S., Trentzsch, M., Richter, R., Moll, P., Fuchs, C., Gehring, O., et al. (2018). "A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond," in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, San Francisco, CA.
- Dutta, S., Chakraborty, W., Gomez, J., Ni, K., Joshi, S., and Datta, S. (2019a). "Energy-Efficient Edge Inference on Multi-Channel Streaming Data," in *Proceedings of the 28nm HKMG FeFET Technology. in 2019 Symposium on VLSI Technology*, Kyoto: IEEE, T38–T39.
- Dutta, S., Saha, A., Panda, P., Chakraborty, W., Gomez, J., Khanna, A., et al. (2019b). "Biologically plausible ferroelectric quasi-leaky integrate and fire neuron," in *Proceedings of the 2019 Symposium on VLSI Technology*, Kyoto: IEEE, T140–T141.
- Faisal, A. A., Selen, L. P. J., and Wolpert, D. M. (2008). Noise in the nervous system. *Nat. Rev. Neurosci.* 9, 292–303. doi: 10.1038/nrn2258
- Gal, Y., and Ghahramani, Z. (2016). "Dropout as a bayesian approximation: representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Machine Learning*, Vol. 48, New York, NY: PMLR, 1050–1059.
- 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
- Gentet, L. J., Stuart, G. J., and Clements, J. D. (2000). Direct measurement of specific membrane capacitance in neurons. *Biophys. J.* 79, 314–320. doi: 10.1016/S0006-3495(00)76293-X
- 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
- Gütig, R. (2014). To spike, or when to spike? *Curr. Opin. Neurobiol.* 25, 134–139. doi: 10.1016/j.conb.2014.01.004
- Hodgkin, A. L., and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* 117, 500–544. doi: 10.1113/jphysiol.1952.sp004764
- Huh, D., and Sejnowski, T. J. (2018). "Gradient descent for spiking neural networks," in *Proceedings of the Advances in Neural Information Processing Systems*, Montréal, QC, 1440–1450.
- Indiveri, G. (2003). "A low-power adaptive integrate-and-fire neuron circuit," in *Proceedings of the - IEEE International Symposium on Circuits and Systems*, Bangkok.
- Indiveri, G., Chicca, E., and Douglas, R. (2006). A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Trans. Neural Networks* 17, 211–221. doi: 10.1109/TNN.2005.860850
- 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
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Networks* 14, 1569–1572. doi: 10.1109/TNN.2003.820440
- Jerry, M., Chen, P. Y., Zhang, J., Sharma, P., Ni, K., Yu, S., et al. (2018a). "Ferroelectric FET analog synapse for acceleration of deep neural network training," in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, San Francisco, CA.
- Jerry, M., Dutta, S., Kazemi, A., Ni, K., Zhang, J., Chen, P.-Y., et al. (2018b). A Ferroelectric field effect transistor based synaptic weight cell. *J. Phys. D: Appl. Phys.* 51:434001. doi: 10.1088/1361-6463/aad6f8
- Jerry, M., Dutta, S., Ni, K., Zhang, J., Sharma, P., Datta, S., et al. (2019). *Ferroelectric FET based Non-Volatile Analog Synaptic Weight Cell*. Notre Dame: University of Notre Dame.
- Jerry, M., Parihar, A., Grisafe, B., Raychowdhury, A., and Datta, S. (2017). "Ultra-low power probabilistic IMT neurons for stochastic sampling machines," in *Proceedings of the IEEE Symposium on VLSI Circuits, Digest of Technical Papers*, Kyoto.
- Joubert, A., Belhadj, B., Temam, O., and Hélot, R. (2012). "Hardware spiking neurons design: Analog or digital?" in *Proceedings of the International Joint Conference on Neural Networks*, Brisbane, QLD.
- Khacef, L., Abderrahmane, N., and Miramond, B. (2018). "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *Proceedings of the International Joint Conference on Neural Networks*, Rio de Janeiro.
- Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Networks* 99, S56–S67. doi: 10.1016/j.neunet.2017.12.005
- Kuzum, D., Jeyasingh, R. G. D., Lee, B., and Wong, H. S. P. (2012). Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing. *Nano Lett.* 12, 2179–2186. doi: 10.1021/nl201040y
- Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539
- Lee, S. R., Kim, Y. B., Chang, M., Kim, K. M., Lee, C. B., Hur, J. H., et al. (2012). "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *Proceedings of the Digest of Technical Papers - Symposium on VLSI Technology*, Honolulu, HI.
- Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128 × 128 120 dB 15 μ s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337
- Liu, Y. H., and Wang, X. J. (2001). Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. *J. Comput. Neurosci.* 10, 25–45. doi: 10.1023/A:1008916026143
- Luo, Y., Wang, P., Peng, X., Sun, X., and Yu, S. (2019). Benchmark of ferroelectric transistor based hybrid precision synapse for neural network accelerator. *IEEE J. Explor. Solid-State Comput. Devices Circuits* 5, 142–150. doi: 10.1109/JXCDC.2019.2925061
- 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* 354, 668–673. doi: 10.1126/science.1254642
- Morie, T., and Amemiya, Y. (1994). An all-analog expandable neural network lsi with on-chip backpropagation learning. *IEEE J. Solid State Circuits* 29, 1086–1093. doi: 10.1109/4.309904
- Mulaosmanovic, H., Chicca, E., Bertele, M., Mikolajick, T., and Slesazeck, S. (2018a). Mimicking biological neurons with a nanoscale ferroelectric transistor. *Nanoscale* 10, 21755–21763. doi: 10.1039/c8nr07135g
- Mulaosmanovic, H., Mikolajick, T., and Slesazeck, S. (2018b). Accumulative polarization reversal in nanoscale ferroelectric transistors. *ACS Appl. Mater. Interfaces* 10, 23997–24002. doi: 10.1021/acsami.8b08967
- Mulaosmanovic, H., Mikolajick, T., and Slesazeck, S. (2018c). Random number generation based on ferroelectric switching. *IEEE Electron Device Lett.* 39, 135–138. doi: 10.1109/LED.2017.2771818
- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595
- Ni, K., Chakraborty, W., Smith, J., Grisafe, B., and Datta, S. (2019a). "Fundamental understanding and control of device-to-device variation," in *Proceedings of the Deeply Scaled Ferroelectric FETs. 2019 Symposium on VLSI Technology*, Kyoto.
- Ni, K., Smith, J. A., Grisafe, B., Rakshit, T., Obradovic, B., Kittl, J. A., et al. (2019b). "SoC logic compatible multi-bit FeMFET weight cell for neuromorphic applications," in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, San Francisco, CA.
- Ni, K., Grisafe, B., Chakraborty, W., Saha, A. K., Dutta, S., Jerry, M., et al. (2018). "In-memory computing primitive for sensor data fusion in 28 nm HKMG FeFET technology," in *Proceedings of the 2018 IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA: IEEE, 11–16.
- 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
- Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *Proceedings of the International Joint Conference on Neural Networks*, Vancouver, BC.
- Park, J., Ha, S., Yu, T., Neftci, E., and Cauwenberghs, G. (2014). "A 65k-neuron 73-Mevents/s 22-pJ/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *Proceedings of the IEEE 2014 Biomedical Circuits and Systems Conference, BioCAS 2014 - Proceedings*, Lausanne.
- Pecevski, D., Buesing, L., and Maass, W. (2011). Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons. *PLoS Comput. Biol.* 7:e1002294. doi: 10.1371/journal.pcbi.1002294

- 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
- Prezioso, M., Merrih-Bayat, F., Hoskins, B. D., Adam, G. C., 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
- 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
- Saha, A. K., Ni, K., Dutta, S., Datta, S., and Gupta, S. (2019). Phase field modeling of domain dynamics and polarization accumulation in ferroelectric HZO. *Appl. Phys. Lett.* 114:202903. doi: 10.1063/1.5092707
- Sengupta, A., Panda, P., Wijesinghe, P., Kim, Y., and Roy, K. (2016). 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
- Sun, X., Wang, P., Ni, K., Datta, S., and Yu, S. (2019). “Exploiting hybrid precision for training and inference: A 2T-1FeFET based analog synaptic weight cell,” in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, San Francisco, CA.
- Trentzsch, M., Flachowsky, S., Richter, R., Paul, J., Reimer, B., Utess, D., et al. (2017). “A 28nm HKMG super low power embedded NVM technology based on ferroelectric FETs,” in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, San Francisco, CA.
- 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
- Wang, R. M., Thakur, C. S., and van Schaik, A. (2018). An FPGA-based massively parallel neuromorphic cortex simulator. *Front. Neurosci.* 12:213. doi: 10.3389/fnins.2018.00213
- Wu, J., Chua, Y., Zhang, M., Yang, Q., Li, G., and Li, H. (2019). “Deep Spiking Neural Network with Spike Count based Learning Rule,” in *Proceedings of the International Joint Conference on Neural Networks*, Budapest.
- Wu, S., Li, G., Chen, F., and Shi, L. (2018). “Training and inference with integers in deep neural networks,” in *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Vancouver, BC.
- Wu, W., Wu, H., Gao, B., Deng, N., Yu, S., and Qian, H. (2017). Improving analog switching in HfOx-based resistive memory with a thermal enhanced layer. *IEEE Electron Device Lett.* 38, 1019–1022. doi: 10.1109/LED.2017.2719161
- Wu, W., Wu, H., Gao, B., Yao, P., Zhang, X., Peng, X., et al. (2018). “A methodology to improve linearity of analog RRAM for neuromorphic computing,” in *Proceedings of the Digest of Technical Papers - Symposium on VLSI Technology*, Honolulu, HI.
- 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
- Yu, S., Chen, P. Y., Cao, Y., Xia, L., Wang, Y., and Wu, H. (2015). “Scaling-up resistive synaptic arrays for neuro-inspired architecture: challenges and prospect,” in *Proceedings of the Technical Digest - International Electron Devices Meeting, IEDM*, Washington, DC.
- 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: 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.

Copyright © 2020 Dutta, Schafer, Gomez, Ni, Joshi and Datta. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Toward Scalable, Efficient, and Accurate Deep Spiking Neural Networks With Backward Residual Connections, Stochastic Softmax, and Hybridization

Priyadarshini Panda^{1*}, Sai Aparna Aketi² and Kaushik Roy²

¹ Department of Electrical Engineering, Yale University, New Haven, CT, United States, ² School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

OPEN ACCESS

Edited by:

Damien Querloz,
Centre National de la Recherche
Scientifique (CNRS), France

Reviewed by:

Timothée Masquelier,
Centre National de la Recherche
Scientifique (CNRS), France
Sumit Bam Shrestha,
Institute for Infocomm Research
(A*STAR), Singapore

*Correspondence:

Priyadarshini Panda
priya.panda@yale.edu

Specialty section:

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

Received: 16 February 2020

Accepted: 26 May 2020

Published: 30 June 2020

Citation:

Panda P, Aketi SA and Roy K (2020)
Toward Scalable, Efficient, and
Accurate Deep Spiking Neural
Networks With Backward Residual
Connections, Stochastic Softmax, and
Hybridization.
Front. Neurosci. 14:653.
doi: 10.3389/fnins.2020.00653

Spiking Neural Networks (SNNs) may offer an energy-efficient alternative for implementing deep learning applications. In recent years, there have been several proposals focused on supervised (conversion, spike-based gradient descent) and unsupervised (spike timing dependent plasticity) training methods to improve the accuracy of SNNs on large-scale tasks. However, each of these methods suffer from *scalability, latency, and accuracy* limitations. In this paper, we propose novel algorithmic techniques of modifying the SNN configuration with *backward residual connections, stochastic softmax, and hybrid artificial-and-spiking neuronal activations* to improve the learning ability of the training methodologies to yield competitive accuracy, while, yielding large efficiency gains over their artificial counterparts. Note, artificial counterparts refer to conventional deep learning/artificial neural networks. Our techniques apply to VGG/Residual architectures, and are compatible with all forms of training methodologies. Our analysis reveals that the proposed solutions yield near state-of-the-art accuracy with significant energy-efficiency and reduced parameter overhead translating to hardware improvements on complex visual recognition tasks, such as, CIFAR10, Imagenet datasets.

Keywords: spiking neural networks, energy-efficiency, backward residual connection, stochastic softmax, hybridization, improved accuracy

1. INTRODUCTION

Neuromorphic computing, specifically, Spiking Neural Networks (SNNs) have become very popular as an energy-efficient alternative for implementing standard artificial intelligence tasks (Indiveri and Horiuchi, 2011; Cao et al., 2015; Panda and Roy, 2016; Sengupta et al., 2016; Pfeiffer and Pfeil, 2018; Roy et al., 2019). Spikes or binary events drive communication and computation in SNNs that not only is close to biological neuronal processing, but also offer the benefit of event-driven hardware operation (Indiveri et al., 2015; Ankit et al., 2017; Roy et al., 2019). This makes them attractive for real-time applications where power consumption and memory bandwidth are important factors. What is lacking, however, is proper training algorithms that can make SNNs perform at par with conventional artificial neural networks (ANNs). Today, there is a plethora of work detailing different

algorithms or learning rules for implementing deep convolutional spiking architectures for complex visual recognition tasks (Masquelier and Thorpe, 2007; Masquelier et al., 2009; O'Connor et al., 2013; Diehl and Cook, 2015; Diehl et al., 2015; Hunsberger and Eliasmith, 2015; Lee et al., 2016, 2018a,b; Panda and Roy, 2016; Mostafa, 2017; Panda et al., 2017; Bellec et al., 2018; Kheradpisheh et al., 2018; Srinivasan et al., 2018; Neftci et al., 2019; Sengupta et al., 2019; Severa et al., 2019; Srinivasan and Roy, 2019). Most algorithmic proposals focus on integrating the discrete or discontinuous spiking behavior of a neuron in a supervised or unsupervised learning rule. All proposals maintain overall sparse network activity (implies low power operation) while improving the accuracy (implies better performance) on image recognition applications [mostly, benchmarked against state-of-the-art datasets like Imagenet (Deng et al., 2009), CIFAR (Krizhevsky and Hinton, unpublished manuscript), MNIST (LeCun et al., 2010)].

Collating the previous works, we can broadly categorize the SNN training methodologies into three types: (1) Conversion from artificial-to-spiking models (Diehl et al., 2015; Sengupta et al., 2019), (2) Approximate Gradient Descent (AGD) based backpropagation with spikes (or accounting temporal events) (Lee et al., 2016; Neftci et al., 2019), and (3) Unsupervised Spike Timing Dependent Plasticity (STDP) based learning (Diehl and Cook, 2015; Srinivasan et al., 2018). Each technique presents some advantages and some disadvantages. While conversion methodology has yielded state-of-the-art accuracies for large datasets like Imagenet on complex architectures [like VGG (Simonyan and Zisserman, 2014), ResNet (He et al., 2016)], the latency incurred to process the rate-coded image¹ is very high (Pfeiffer and Pfeil, 2018; Lee et al., 2019; Sengupta et al., 2019). AGD training addresses the latency concerns yielding $\sim 10 - 15\times$ benefits as compared to the conversion (Bellec et al., 2018; Lee et al., 2019; Neftci et al., 2019). However, AGD still lags behind conversion in terms of accuracy for larger and complex tasks. The unsupervised STDP training, while being attractive for real-time hardware implementation on several emerging and non-von Neumann architectures (Pérez-Carrasco et al., 2010; Linares-Barranco et al., 2011; Ankit et al., 2017; Sengupta and Roy, 2017; van de Burgt et al., 2017; Wang et al., 2017), also suffers from accuracy/scalability deficiencies.

From the above discussion, we can gather that addressing *Scalability, Latency, and Accuracy* issues are key toward achieving successful SNN methodologies. In this paper, we precisely address each of these issues through the lens of *network architecture modification, softmax classifier adaptation, and network hybridization with a mix of Rectified Linear Unit/ReLU (or ANN-like) and Leaky-Integrate-and-Fire (or SNN-like) neuronal activations in different layers*.

¹SNNs process event data obtained with rate or temporal coding instead of real-valued pixel data. Rate coding is widely used for SNN applications, where, a real-valued pixel data is converted to a Poisson-distribution based spike train with the spiking frequency proportional to the pixel value (Diehl and Cook, 2015). That is, high valued pixels output more spikes and vice-versa.

2. RELATED WORK, MOTIVATION, AND CONTRIBUTIONS

2.1. Addressing Scalability With Backward Residual Connections

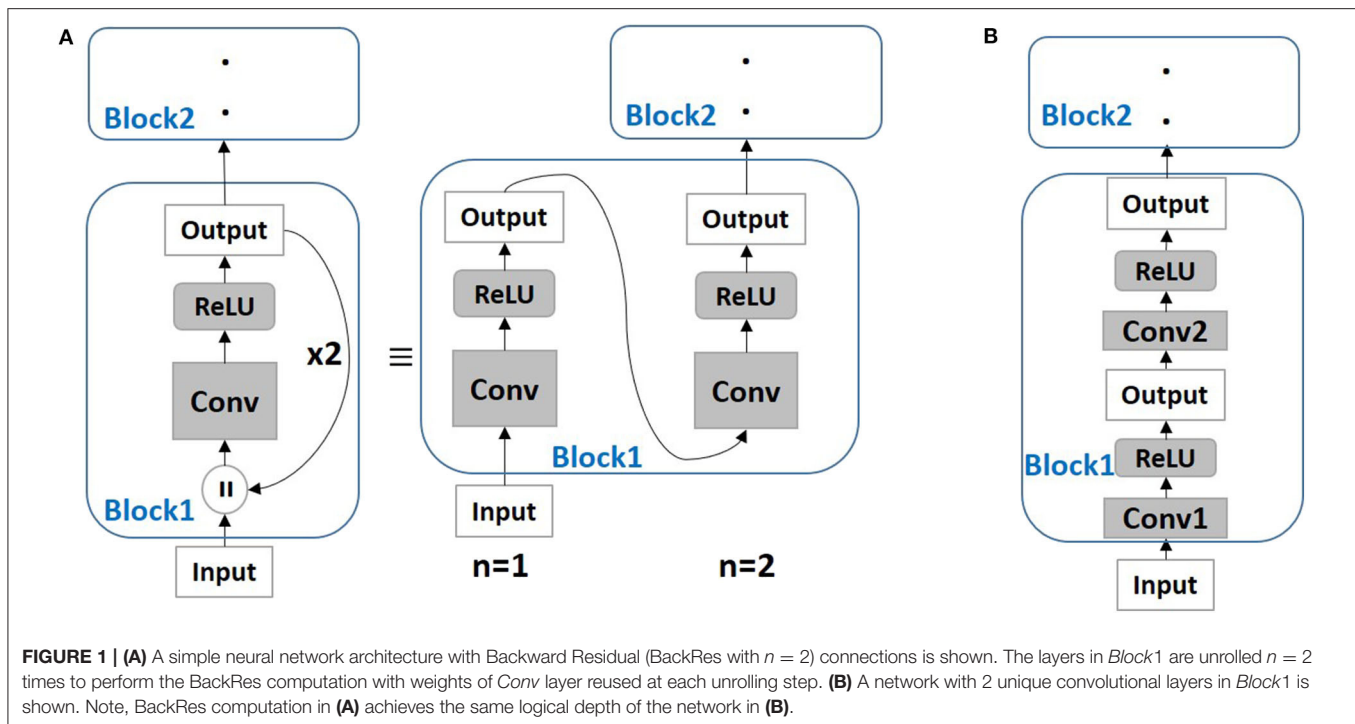
Scalability limitations of STDP/AGD approaches arises from their depth incompatibility with deep convolutional networks which are necessary for achieving competitive accuracies. SNNs forward propagate spiking information and thus require sufficient spike activity across all layers of a deep network to conduct training. However, previous works have shown that spiking activity decreases drastically for deeper layers of a network (that we define as *vanishing spike propagation*), thereby, causing training issues for networks with large number of layers (Masquelier et al., 2009; Diehl et al., 2015; Lee et al., 2016, 2018b; Panda and Roy, 2016; Kheradpisheh et al., 2018; Srinivasan et al., 2018).

From ANN literature, it is known that depth is key to achieving improved accuracy for image recognition applications (LeCun et al., 2015; Szegedy et al., 2015). Then, the question arises, *can we modify the spiking network architecture to be less deep without compromising accuracy?* Kubilius et al. (2018) proposed Core Object Recognition or CORnet models (with what we term as *backward residual connections*) that transform deep feedforward ANN models into shallow recurrent models. **Figure 1** illustrates the Backward Residual (BackRes) block architecture. It is similar to that of a recurrent network unrolled over time with weights shared over repeated computations of the output. Specifically, the computations in *Block1* are performed twice before processing *Block2*. For $n = 1$, *Block1* processes original input information, while, for $n = 2$, the same *Block1* with repeated weights processes the output from previous step. Note, the original input is processed only once for $n = 1$. For $n > 1$, the block processes its output from the previous step. Essentially, BackRes connections enable a network to achieve similar logical depth as that of a deep feedforward network without introducing additional layers. The 1-convolutional layer block in **Figure 1A** achieves the logical depth of a 2-convolutional layer block as shown in **Figure 1B** and is expected to achieve near iso-accuracy with that of the 2-convolutional layer block². The BackRes connection brings two key advantages: (1) Reduction in the total number of parameters since we are reusing the same weights over multiple steps of unrolling, (2) Diversification of gradient update for each unrolled step due to different input-output combinations.

2.1.1. Our Contribution

We utilize BackRes connections and the diversified gradients to enable training of logically deep SNN models with AGD or STDP that otherwise cannot be trained (with multiple layers) due to vanishing spike propagation. Further, we show that converting a deep ANN (with BackRes blocks) into a

²There is a limit to which BackRes compensates for depth diversity with iso-accuracy. VGG2x8 network with 2 convolutional layers unrolled 8 times may suffer accuracy loss as compared to a VGG16 network with 16 convolutional layers. But, VGG2x4 may yield near iso-accuracy as VGG8. Note, VGG2x4 and VGG8 have same logical depth of 8 convolutional layers.



deep SNN necessitates the use of multiple threshold-spiking neurons per BackRes block to achieve lossless conversion. We also demonstrate that BackRes SNN models (say, VGG2x4) yield both lower memory complexity (proportional to number of weights/parameters) and sparser network activity with decreased computational overhead (proportional to total inference energy) as compared to a deep architecture (say, VGG8) of similar logical depth across different SNN training methodologies.

2.2. Addressing Latency With Stochastic Softmax (Stochmax)

In order to incur minimal loss during pixel-to-spike conversion with rate coding¹ (generally, used in all SNN experiments), the number of time steps of the spike train has to sufficiently large. This, in turn, increases the latency of computation. Decreasing the latency implies larger loss in image-to-spike conversion that can result in lower accuracy.

Across all SNN training methodologies, the final classifier or output layer which yields the prediction result is usually a softmax layer similar to that of an ANN. It is general practice, in SNN implementation, to collect all the accumulated spiking activity over a given time duration from the penultimate layer of a deep SNN and feed it to a softmax layer that calculates the loss and prediction based on the integrated spike information (Masquelier and Thorpe, 2007; Lee et al., 2016, 2019). While the softmax classifier based training has produced competitive results, the latency incurred still is significantly high. The question that arises here is, “Can we compensate for reduced latency (or, higher loss during image-to-spike conversion) by improving the learning

capability of the SNN by augmenting the softmax functionality?”

Lee H. B. et al. (2018) proposed a stochastic version of a softmax function (*stochmax*) that drops irrelevant (non-target) classes with adaptive dropout probabilities to obtain improved accuracy in ANN implementations. Stochmax can be viewed as a stochastic attention mechanism, where, the classification process at each training iteration selects a subset of classes that the network has to attend to for discriminating against other false classes. For instance, while training for a *cat* instance, it is useful to train the model with more focus on discriminating against confusing classes, such as, *jaguar*, *tiger* instead of orthogonal classes like *truck*, *whale*. Softmax, on the other hand, collectively optimizes the model for target class (*cat*) against all remaining classes (*jaguar*, *tiger*, *truck*, *whale*) in an equally weighted manner, thereby, not involving attentive discrimination.

2.2.1. Our Contribution

Given that stochmax improves intrinsic discrimination capability, we utilized this stochastic regularization effect to decrease the training/inference latency in SNN frameworks. We show how standard AGD can be integrated with stochmax classifier functionality to learn deep SNNs. Our analysis yields that deep SNNs of 3–4 layers trained with stochmax yield higher accuracy at lower latency than softmax baselines (for AGD training).

2.3. Addressing Accuracy With Network Hybridization

It is evident that accuracy loss due to *vanishing spike propagation* and *input pixel-to-spike coding* are innate properties of SNN design that can be addressed to certain extent, but, cannot be

completely eliminated. In order to achieve competitive accuracy as that of an ANN, we believe that taking a hybrid approach with a partly-artificial-and-partly-spiking neural architecture will be most beneficial.

2.3.1. Our Contribution

We demonstrate a hybrid neural architecture for AGD training methodologies. In case of AGD, since the training is performed end-to-end in a deep network, vanishing spike-propagation becomes a limiting factor to achieve high accuracy. To address this, we use ReLU based neurons in the initial layers and have spiking leaky-integrate-and-fire neurons in the latter layers and perform end-to-end AGD backpropagation. In this scheme, the idea is to extract relevant activity from the input in the initial layers with ReLU neurons. This allows the spiking neurons in latter layers to optimize the loss function and backpropagate gradients appropriately based on relevant information extracted from the input without any information loss.

Finally, we show the combined benefits of incorporating BackRes connections with stochmax classifiers and network hybridization across different SNN training methodologies and show latency, accuracy, and compute-efficiency gains. Through this work, our goal is to communicate good practices for deploying SNN frameworks that yield competitive performance and efficiency as compared to corresponding ANN counterparts.

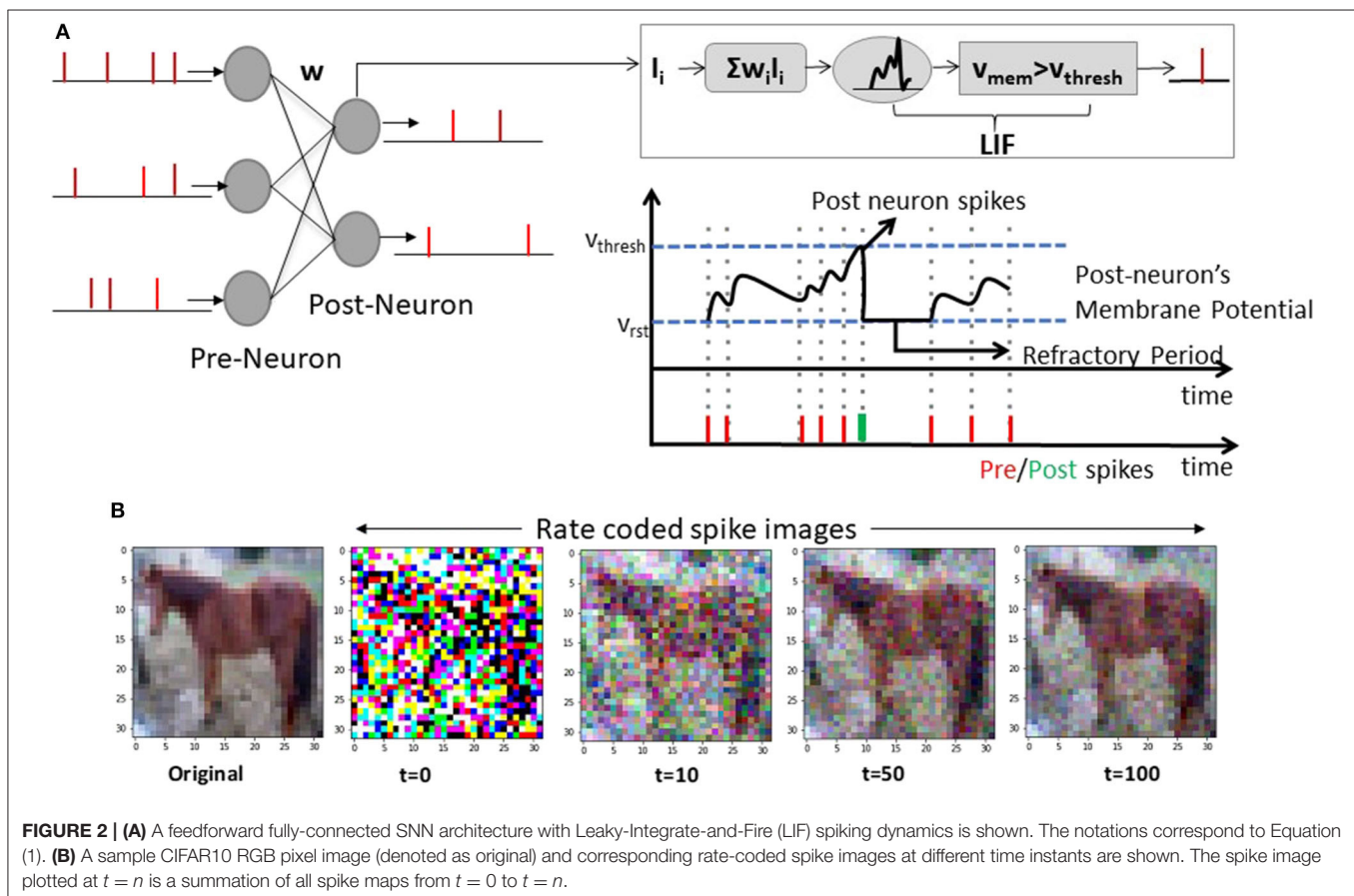
3. SNN: BACKGROUND AND FUNDAMENTALS

3.1. Input and Neuron Representation

Figure 2A illustrates a basic spiking network architecture with Leaky-Integrate-and-Fire (LIF) neurons processing rate-coded inputs¹. It is evident from Figure 2B that converting pixel values to binarized spike data $\{1: \text{spike}, 0: \text{no spike}\}$ in the temporal domain preserves the integrity of the image over several time steps. The dynamics of a LIF spiking neuron is given by

$$\tau \frac{dv_{mem}}{dt} = -v_{mem} + \sum_i I_i w_i \quad (1)$$

The membrane potential v_{mem} integrates incoming spikes I_i through weights w_i and leaks (with time constant τ) whenever it does not receive a spike. The neuron outputs a spike event when v_{mem} crosses certain threshold v_{thresh} . Refractory period ensues after spike generation during which the post-neuron's membrane potential is not affected. In some cases, Integrate-and-Fire (IF) neurons are also used where leak value is 0 for simplicity in simulations/hardware implementations. Note, while Figure 2 illustrates a fully-connected network, SNNs can be constructed with a convolutional hierarchy comprising multiple layers. For the sake of notation, we will refer to networks with real-valued



computations/ReLU neurons as ANNs and networks with spike-based computations/LIF or IF neurons as SNNs.

3.2. Training Methodology

3.2.1. Conversion From ANN-to-SNN

To achieve higher accuracy with SNNs, a promising approach has been to convert ANNs trained with standard backpropagation into spiking versions. Fundamentally, the goal here is to match the input-output mapping function of the trained ANN to that of the SNN. Recent works (Diehl et al., 2015; Sengupta et al., 2019) have proposed weight normalization and threshold balancing methods in order to obtain minimal loss in accuracy during the conversion process. In this work, we use the threshold balancing method (Sengupta et al., 2019) that yields almost zero-loss ANN-to-SNN conversion performance for deep VGG/ResNet-like architectures on complex Imagenet dataset.

In threshold balancing, after obtaining the trained ANN, the first step is to generate a Poisson spike train corresponding to the entire training dataset for a large simulation duration or time period (generally, 2,000–2,500 time steps). The Poisson spike train allows us to record the maximum summation of weighted spike input ($\sum_j w_{ij} X_j(t)$) received by the first layer of the ANN. v_{thresh} value for the first layer is then set to the maximum summation value. After the threshold for the first layer is set, the network is again fed the input data to obtain a spike-train at the first layer, which serves as the input spike-stream for the second layer of the network. This process of generating spike train and setting v_{thresh} value is repeated for all layers of the network. Note, the weights during this balancing process remain unchanged. For more details on this technique (please see Sengupta et al., 2019).

While conversion approach yields high accuracy, the computation cost is large due to high latency in processing. Reducing the time period from 2,000 to 100/10 time steps causes large decline in accuracy as v_{thresh} balancing fails to match the output rate of SNN to that of ANN. Note, the accuracy of an SNN in conversion case is bounded by the accuracy of the corresponding ANN.

3.2.2. Approximate Gradient Descent (AGD)

The thresholding functionality in the spiking neuron yields a discontinuous/non-differentiable functionality making it incompatible with gradient-descent based learning methods. Consequently, several training methodologies have been proposed to incorporate the temporal statistics of SNNs and overcome the gradient descent challenges (O'Connor et al., 2013; Lee et al., 2016, 2018b; Panda and Roy, 2016; Bellec et al., 2018; Neftci et al., 2019). The main idea is to approximate the spiking neuron functionality with a continuously differentiable model or use surrogate gradients as a relaxed version of the real gradients to conduct gradient descent training. In our work, we use the surrogate gradient approach proposed in Neftci et al. (2019).

In Neftci et al. (2019), the authors showed that temporal statistics incorporated in SNN computations can be implemented as a recurrent neural network computation graph (in, PyTorch, Tensorflow; Abadi et al., 2016 frameworks) that can be unrolled to conduct Backpropagation Through Time (BPTT) (Werbos et al., 1990). The authors in Neftci et al. (2019) also showed

that using LIF computations in the forward propagation and surrogate gradient derivatives during backpropagation allows SNNs (of moderate depth) to be efficiently trained end-to-end. Using a recurrent computational graph enables the use of BPTT for appropriately assigning the gradients with chain rule in the temporal SNN computations. Here, for a given SNN, rate coded input spike trains are presented and the output spiking activity at the final layer (which is usually a softmax classifier) is monitored for a given time period. At the end of the time period, the loss from the final softmax layer is calculated and corresponding gradients are backpropagated through the unrolled SNN computation graph.

Figure 3A illustrates the SNN computational graph. From an implementation perspective, we can write the dynamics of an LIF neuron in discrete time as

$$V_{mem_i}[t+1] = \alpha V_{mem_i}[t] + I_i[t] \quad (2)$$

$$I_i[t] = \sum_j w_{ij} S_j[t] \quad (3)$$

Here, the output spike train S_i of neuron i at time step t is a non-linear function of membrane potential $S_i[t] \equiv \Theta(V_{mem_i}[t] - v_{thresh})$ where Θ is the Heaviside step function and v_{thresh} is the firing threshold. I_i is the net input current and $\alpha = \exp(-\Delta t / \tau_{mem})$ is the decay constant (typically in the range $\{0.95, 0.99\}$). During backpropagation, the derivative of $S(V_{mem}(t)) = \Theta(V_{mem}(t) - v_{thresh})$ is zero everywhere except at $V_{mem} = v_{thresh}$ where it is not defined. This all-or-nothing behavior of spiking neurons stops gradient from flowing through chain rule making it difficult to perform gradient descent. We approximate the gradient using surrogate derivatives for Θ following (Bellec et al., 2018; Neftci et al., 2019) as

$$\frac{dS[t]}{dV_{mem}[t]} = \gamma \max\{0, 1 - \frac{V_{mem}[t] - v_{thresh}}{v_{thresh}}\} \quad (4)$$

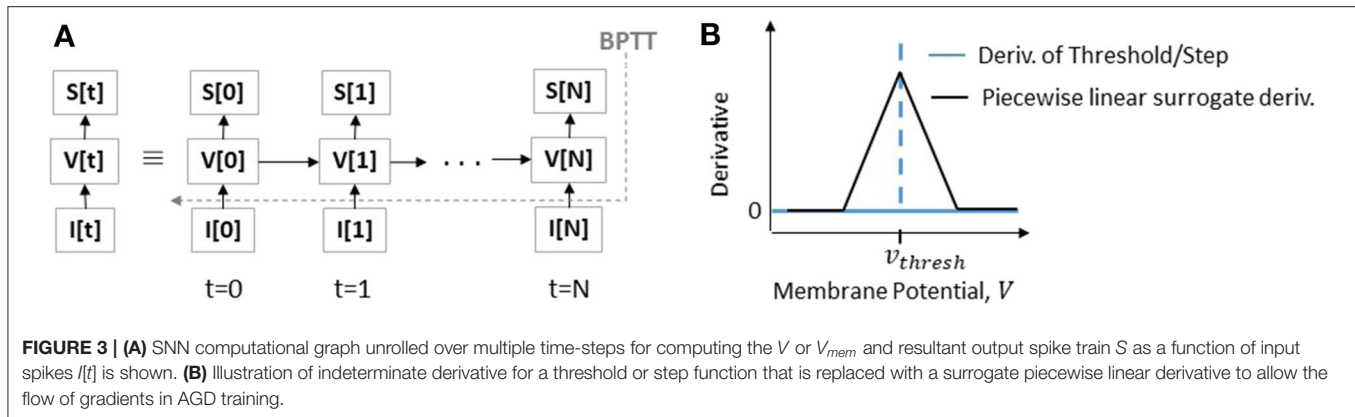
where γ is a damping factor (set to 0.3) that yields stable performance during BPTT. As shown in Figure 3B, using a surrogate gradient (Equation 4) now replaces a zero derivative with an approximate linear function. For more details and insights on surrogate gradient descent training (please see Bellec et al., 2018; Neftci et al., 2019). For convenience in notation, we will use AGD to refer to surrogate descent training in the remainder of the paper.

Using end-to-end training with spiking computations enables us to lower the computation time period to 50–100 time steps. However, these methods are limited in terms of accuracy/performance and are also not suitable for training very deep networks.

3.2.3. Unsupervised STDP Learning

STDP is a correlation based learning rule which modulates the weight between two neurons based on the correlation between pre- and post-neuronal spikes. In this work, we use a variant of the STDP model used in Diehl and Cook (2015), Srinivasan et al. (2018), and Srinivasan and Roy (2019) described as

$$\Delta w_{STDP} = \eta \times (e^{-\frac{t_{post} - t_{pre}}{\tau}} - STDP_{offset}) \quad (5)$$



where Δw_{STDP} is the weight update, η is the learning rate, t_{post}, t_{pre} are the time instants of post- and pre-neuronal spikes, τ is the STDP time constant. Essentially, the weight is increased if a pre-neuron triggers a post-neuron to fire within a time period specified by the $STDP_{offset}$ implying strong correlation. If the spike timing difference is large between the pre- and post-neurons, the weight is decreased. In Srinivasan and Roy (2019), the authors implemented a mini-batch version of STDP training for training convolutional SNNs in a layerwise manner. For training the weight kernels of the convolutional layers shared between the input and output maps, the pre-/post-spike timing differences are averaged across a given mini-batch and corresponding STDP updates are performed. In this work, we perform mini-batch training as in Lee et al. (2018b) and Srinivasan and Roy (2019). We also use the uniform threshold adaptation and dropout scheme following (Lee et al., 2018b; Srinivasan et al., 2018; Srinivasan and Roy, 2019) to ensure competitive learning with STDP. For more information on the learning rule (please see Lee et al., 2018b; Srinivasan et al., 2018).

Generally, a network trained with layerwise STDP (for convolutional layers) is appended with a classifier (separately trained with backpropagation) to perform final prediction. The authors in Srinivasan and Roy (2019) showed that unsupervised STDP learning (even with binary/probabilistic weight updates) of a deep SNN, appended with a fully-connected layer of ReLU neurons, yields reasonable accuracy. However, similar to AGD, layerwise STDP training is not scalable and yields restrictive performance for deep multi-layered SNNs.

4. SNNs WITH BACKRES CONNECTIONS

BackRes allows a model to perform complex computation over multiple logical depth by means of repeated unrolling. From Figure 1, it appears that the number of output and input channels in a BackRes block need to be equal for consistency. However, given a BackRes block with 64 input channels and 128 output channels (say, VGG2x4 network), one can randomly drop 64 channels from the output during unrolled computations. Selecting top-64 channels with maximal activity, or averaging the response of 128 channels into 64 also yields similar accuracy as that of a baseline network (VGG8).

For convenience, in our experiments, we use models with same input/output channels and convert them to BackRes blocks. Next, we discuss how to integrate BackRes connection for different SNN training methodologies.

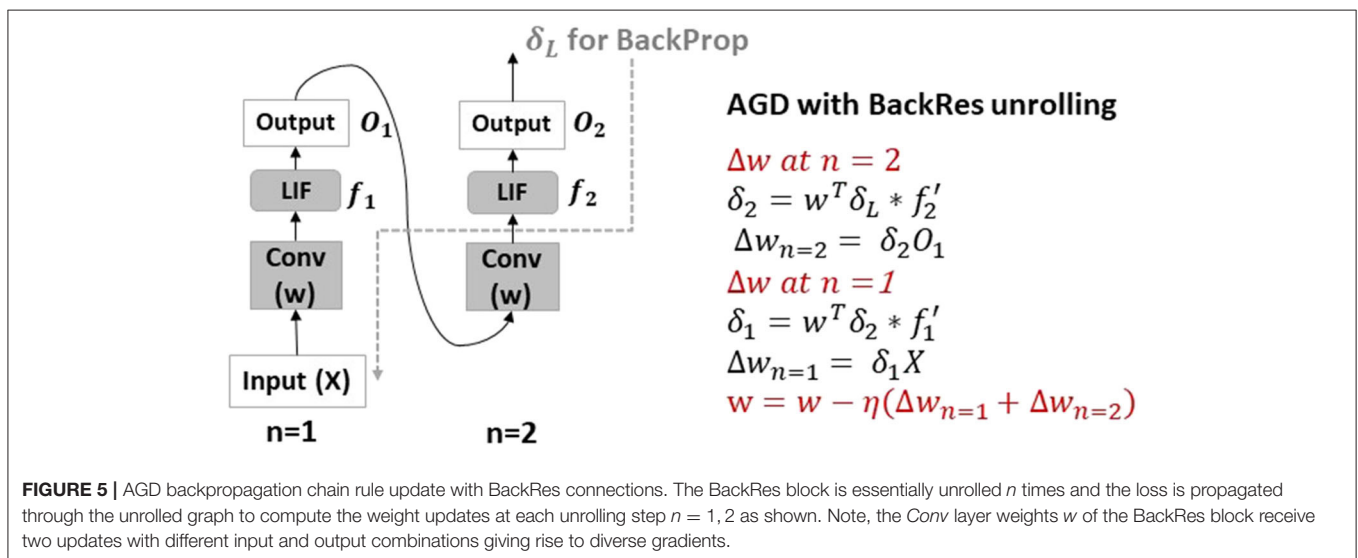
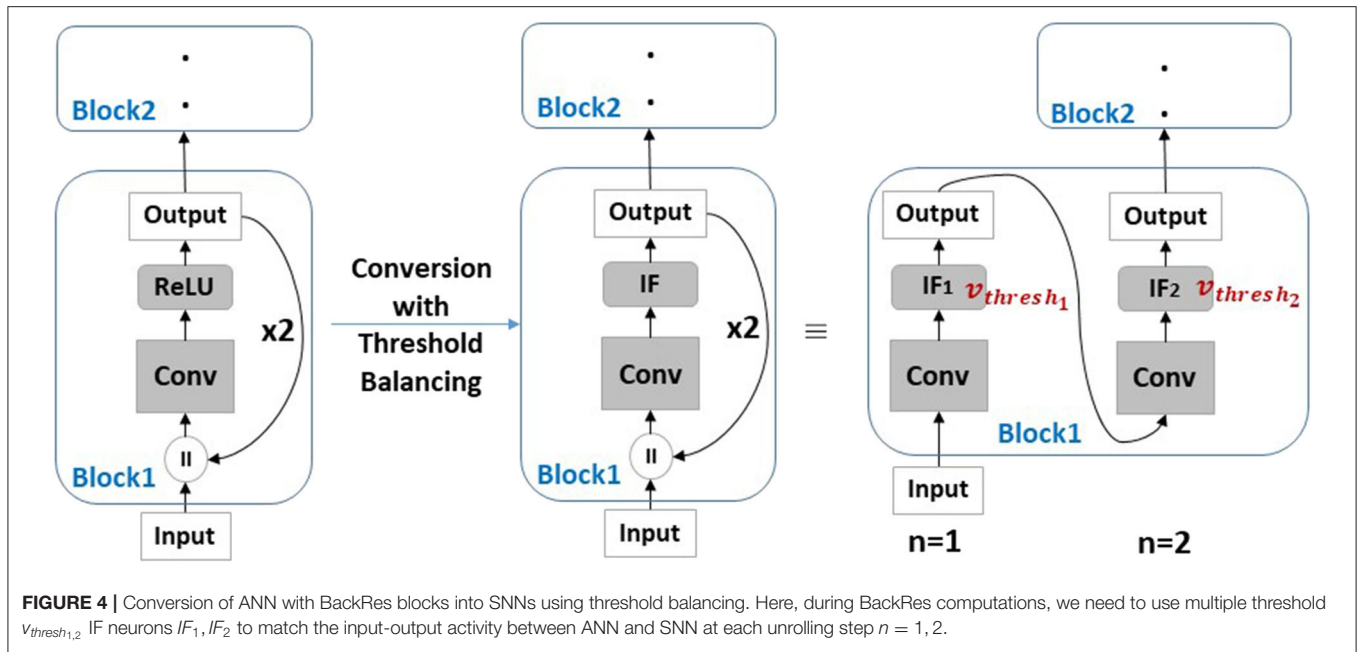
4.1. Conversion

In this methodology, SNN is constructed from a trained ANN. Hence, the ANN has to incorporate BackRes connections with repeated ReLU computations (similar to Figure 1) which then need to be appropriately matched to spiking neuronal rates. Figure 4 illustrates the conversion from ReLU to IF neurons. Here, since unrolling each time yields a different output rate, we need to ensure that we use multiple threshold IF neurons where IF_1 with threshold v_{thresh_1} is activated for $n = 1$ and IF_2 with threshold v_{thresh_2} for $n = 2$. Thus, the number of thresholds v_{thresh_n} will be equal to the number of unrolling steps n . During threshold balancing for conversion (see section 3.2.1), we need to set the thresholds for each layer as well as each step of unrolling within a layer separately. Interestingly, we find that v_{thresh} increases with n , i.e., $v_{thresh_1} < v_{thresh_2} \dots < v_{thresh_n}$. Increasing threshold implies lesser spiking activity with each unrolling which reduces the overall energy cost (results shown in section 8.1).

4.2. AGD Training

In AGD training, an SNN is trained end-to-end with the loss calculated at the output layer using surrogate gradient descent on LIF neurons. The thresholds of all neurons are set to a user-defined value at the beginning of training and remain constant throughout the learning process. The weight updates inherently account for the balanced spiking activity given the set thresholds. Adding BackRes blocks in this case will be similar to training a recurrent model with unrolled computation, that is treating the BackRes block as a feedforward network of n layers. During backpropagation, the gradients are passed through the unrolled graph of the BackRes block, where, the same weights w are updated n times.

Figure 5 illustrates the backpropagation chain rule update. It is worth mentioning that the LIF activity with every unrolling varies, that eventually affects the weight update value at each step. As in conversion, we find that networks with BackRes



blocks and shared weights (say, VGG2x2) generally have lower spiking activity than equivalent depth baseline network with separate layers (say, VGG4), yielding energy improvements. This implies that the repeated computation with unrolling gives rise to diverse activity that can possibly model diverse features, thereby, allowing the network to learn even with lesser depth. Note, the BackRes network and the baseline network have same v_{thresh} through all layers when trained with AGD. Further, AGD training has scalability limitations. For instance, a seven-layered VGG network fails to learn with end-to-end surrogate gradient descent. However, a network with BackRes blocks with real depth of five layers and logical depth of seven layers can now be easily trained and in fact yields competitive accuracy (results shown in section 8.1).

4.3. STDP Training

SNNs learnt with STDP are trained layerwise in an iterative manner. Generally, in one iteration of training (comprising of T time-steps or 1 time period of input presentation), a layer's weights are updated k times ($k \leq T$) depending upon the total spike activity in the pre-/post-layer maps and spiking correlation (as per Equation 5). Since BackRes performs n repeated computations of a single layer, in this case, we make $k \times n$ weight updates for the given layer in each iteration of STDP training. From Figure 5, we can gather that the pre-/post-correlation at $n = 1$ unrolling step will correspond to input X and Conv layer's output that will determine its weight updates. For $n = 2$, the Conv layer's output from previous step will serve as pre-spiking activity based on which the weights are

updated again. Similar to AGD training, the overall activity at the output of *Conv* changes with n which diversifies and improves the capability of the network to learn better. We also find reduced energy cost and better scalability toward large logical depth networks that otherwise (with real depth) could not be trained in a layerwise manner (results shown in section 8.1).

5. SNNS WITH STOCHMAX

Stochmax as noted earlier is a stochastic version of a softmax function that allows a network to discriminate better by focusing or giving importance to confusing classes. A softmax classifier is defined as

$$p(y|x; \theta) = \frac{\exp(o_t(x; \theta))}{\sum_k \exp(o_k(x; \theta))} \quad (6)$$

where t is the target label for input x , k is the number of classes, and $o(x; \theta) = W^T h + b$, $\theta = \{W, b\}$ is the logits score generated from the last feature vector $h = NN(x; \omega)$ of a neural network $NN(\cdot)$ parameterized by ω . With Stochmax, the objective is to randomly drop out classes in the training phase with a motivation of learning an ensemble of classifiers in a single training iteration. From Equation (6), we can see that making $\exp(o_k) = 0$ drops class k completely even eliminating its gradients for backpropagation. Following this, Stochmax is defined as:

$$z_k|x \sim \text{Ber}(z_k; \rho_k(x; \theta)),$$

$$p(y|x, z; \theta, \psi) = \frac{(z_t + \epsilon) \exp(o_t(x; \theta))}{\sum_k (z_k + \epsilon) \exp(o_k(x; \theta))} \quad (7)$$

Here, we drop out classes with a probability $(1 - \rho_k)$ based on Bernoulli (*Ber*) trials. Further, to encode meaningful correlations in the probabilities ρ_k , we learn the probabilities as an output of the neural network which takes last feature vector h as input and outputs a sigmoidal value $\rho(x; \psi) = \sigma(W^T \psi + b_\psi)$, $\psi = \{W_\psi, b_\psi\}$. By learning ψ , we expect that highly correlated classes can be dropped or retained together. In essence, by dropping classes, we let the network learn on different *sub-problems* at each iteration. In SNN implementations, we replace the softmax classifier (Equation 6) with a Stochmax function (Equation 7) at the output. Generally, the classifier layer is a non-spiking layer which receives accumulated input from the previous spiking layer h integrated over the T time-steps per training iteration. The loss is then calculated from stochmax output which is used to calculate the gradients and perform weight updates.

It is evident that AGD training where the loss function at the classifier is used to update the weights at all layers of a deep SNN will be affected by this softmax-to-stochmax replacement. We find that this attentive discrimination that implicitly models many classifiers (providing different decision boundaries) per training iteration allows an SNN to be trained even with lower latency (or lesser time steps per training iteration or input presentation) while yielding high accuracy. Lower latency implies that pixel-to-spike input coding with Poisson rate will incur more loss. However, the deficit of the input coding gets rectified with improved classification.

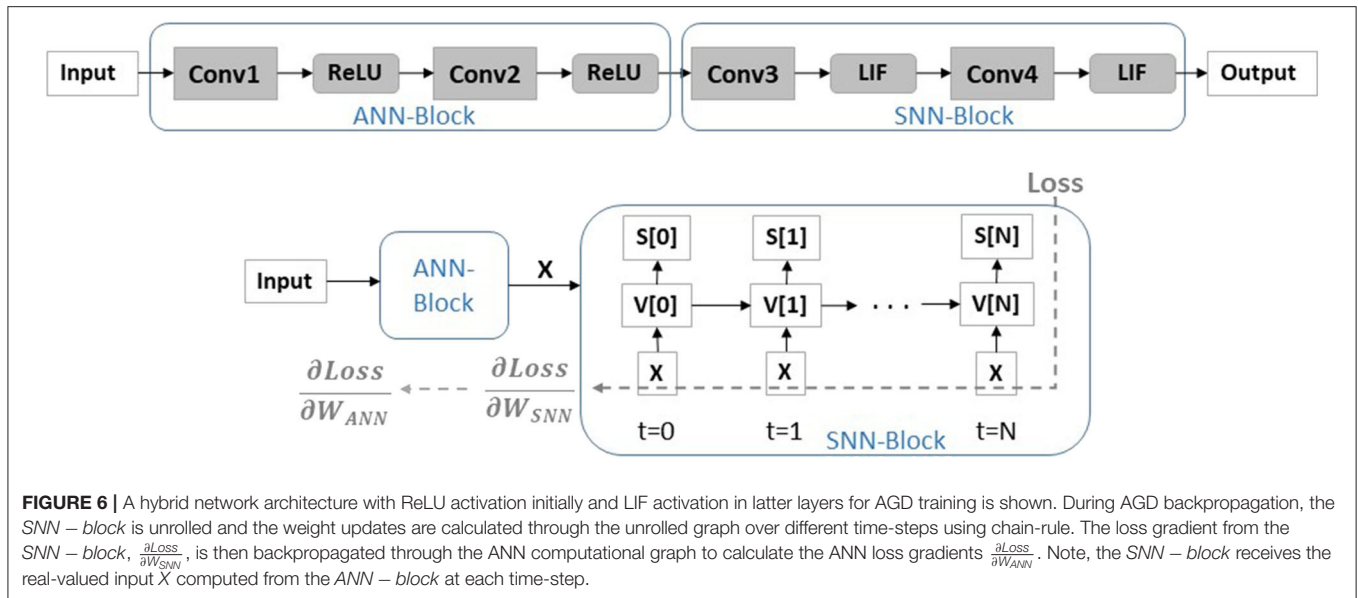
In Conversion, an ANN is trained separately and is completely dissociated from the spiking statistics. STDP, on similar lines, has spike-based training of intermediate feature extractor layers. The final classifier layers (which are separately trained) are appended to the STDP-trained layers and again do not influence the weight or activity learnt in the previous layers. Thus, while Stochmax classifier inclusion slightly improves the accuracy of both conversion/STDP methods, they remain unaffected from a latency perspective.

6. SNNS WITH HYBRID RELU-AND-LIF NEURONS

The objective with a partly-artificial-and-partly-spiking neural architecture is to achieve improved accuracy. For artificial-to-spiking conversion methodology, since training is performed using ReLU neuronal units and inference with spiking integrate-and-fire neurons, network hybridization is not necessary and will not add to the overall accuracy. Most works on STDP learning use hybrid network architecture where STDP is used to perform feature extraction with greedy layer-wise training of the convolutional layers of a deep network. Then, a one-layer fully connected ANN (with ReLU neurons) is appended to the STDP trained layers to perform final classification. However, STDP is limited in its capability to extract specific features from the input that are key for classification. We find that strengthening the ANN hierarchy of an STDP-trained SNN (either with Stochmax or deepening the ANN with multiple layers) yields significant improvement in accuracy.

In AGD, since learning is performed end-to-end vanishing spike-propagation restricts the training of a deep many-layered network. For instance, a VGG7 network fails to train with AGD. In fact, even with residual or skip connections that leads to a ResNet7-like architecture, the model is difficult to train. BackRes connections are potential solutions for training logically deep networks. However, to achieve better accuracy for deeper many-layered networks, there is a need to hybridize the layers of the network with ReLU and LIF neurons.

Figure 6 illustrates the hybrid network configuration. We have ReLU neurons in initial layers and temporal LIF neurons in latter layers. During forward propagation, the input processed through the ANN — *block* is then propagated through the SNN — *block* unrolled over different time-steps as a recurrent computational graph to calculate the loss at the final output layer (that can be softmax/stochmax function). In the backward phase, the gradient of loss is propagated through the recurrent graph updating the weights of the SNN block with surrogate linear approximation of the LIF functionality corresponding to activity at each time step. The loss gradient calculated through BPTT are then passed through the ANN-block (which calculates the weight updates in ANN with standard chain rule). It is worth mentioning that setting up a hybrid network in a framework like PyTorch automatically performs recurrent graph unrolling for SNN-block and standard feedforward graph for ANN-block and enables appropriate



gradient calculation and weight updates. We would also like to note that we feed in the output of the ANN-block as it is (without any rate-coding) to the SNN-block. That is, the unrolled SNN graph at each time-step receives the same real-valued input X . We find that processing X instead of rate-coded $X[t]$ yields higher accuracy at nearly-same energy or computation cost.

Note, there has been recent works (Pei et al., 2019; Voelker et al., 2020) that also portray hybrid SNN-ANN implementations. We would like to clarify that our partly-artificial-and-partly-spiking hybrid neural network implementation is very different from Pei et al. (2019) and Voelker et al. (2020). In Pei et al. (2019), the authors propose a hybrid accelerator that can operate both an SNN and ANN. In their hybrid-mode network implementation, the hybrid layer used “SNN-input and ANN-output” cores to integrate SNN spikes and generate ANN signals (high-precision intermediate membrane potentials), and then used “ANN-input and SNN-output” cores to accumulate these ANN signals and fire SNN spikes again. On similar lines, the hybrid implementation proposed in Pei et al. (2019) is completely different from our technique. In Voelker et al. (2020), the authors smoothly interpolate between ANN (i.e., 32-bit activities) and SNN (i.e., 1-bit activities) regimes. The key idea is to interpret spiking neurons as one-bit quantizers that diffuse their quantization error across future time-steps. In other words, the entire network has neuronal activation initialized as a temporally diffused quantizer and during the training period, the quantizer is scaled. Both the approaches are very different from our proposed hybrid ANN-SNN training scheme. In our hybrid scheme, we essentially initialize a hybrid ANN-SNN multi-layered network wherein, certain layers have standard ReLU neuronal activation and certain layers have IF neuronal activation that is then trained with appropriate SNN layer unrolling for proper gradient assignment as shown in **Figure 6**.

7. EXPERIMENTS

We conduct a series of experiments for each optimization scheme, primarily, using CIFAR10 and Imagenet data on VGG and ResNet architectures of different depths detailing the advantages and limitation of each approach. We implemented all SNN models in PyTorch and used the same hyperparameters (such as, leak time constant, v_{thresh} value, input spike rate etc.) as used in Sengupta et al. (2019), Neftci et al. (2019), Srinivasan and Roy (2019) for conversion, surrogate gradient descent, and STDP training, respectively. In all experiments, we measure the accuracy, latency, energy or total compute cost, and total parameters for a given SNN implementation and compare it to the baseline ANN counterpart. Latency is measured as total number of time-steps required to perform an inference for one input. In case of ANN, latency during inference is 1, while, SNN latency is the total number of time-steps T over which an input is presented to the network. Note, in all our experiments, all ANNs and SNNs are trained for different number of epochs/iterations until maximum accuracy is achieved in each case.

The total compute cost is measured as total number of floating point operations (FLOPS) which is roughly equivalent to the number of multiply-and-accumulate (MAC) or dot product operations performed per inference per input (Han et al., 2015a,b). In case of SNN, since the computation is performed over binary events, only accumulate (AC) operations are required to perform the dot product (without any multiplier). Thus, SNN /ANN FLOPS count will consider AC/MAC operations, respectively. For a particular convolutional layer of an ANN/SNN, with N input channels, M output channels, input map size $I \times I$, weight kernel size $k \times k$ and output size $O \times O$, total FLOPS count for ANN/SNN is

$$FLOPS_{ANN} = O^2 * N * k^2 * M \quad (8)$$

$$FLOPS_{SNN} = O^2 * N * k^2 * M * S_A \quad (9)$$

TABLE 1 | Energy table for 45 nm CMOS process.

Operation	Energy (pJ)
32-b MULT Int	3.1
32-b ADD Int	0.1
32-b MAC Int	3.2 (E_{MAC})
32-b AC Int	0.1 (E_{AC})

Note, $FLOPS_{SNN}$ in Equation (9) is calculated per time-step and considers the net spiking activity (S_A) that is the total number of firing neurons per layer. In general, $S_A \ll 1$ in an SNN on account of sparse event-driven activity, whereas, in ANNs $S_A = 1$. For energy calculation, we specify each MAC or AC operation at the register transfer logic (RTL) level for 45 nm CMOS technology (Han et al., 2015b). Considering 32-bit weight values, the energy consumption for a 32-bit integer MAC/AC operation (E_{MAC}, E_{AC}) is shown in **Table 1**. Total inference energy E for ANN/SNN considering FLOPS count across all N layers of a network is defined as

$$E_{ANN} = \left(\sum_{i=1}^N FLOPS_{ANN} \right) * E_{MAC} \quad (10)$$

$$E_{SNN} = \left(\sum_{i=1}^N FLOPS_{SNN} \right) * E_{AC} * T \quad (11)$$

For SNN, the energy calculation considers the latency incurred as the rate-coded input spike train has to be presented over T time-steps to yield the final prediction result. Note, this calculation is a rather rough estimate which does not take into account memory access energy and other hardware architectural aspects such as input-sharing or weight-sharing. Given, that memory access energy remains same irrespective of SNN or ANN network topology, the overall *Energy-Efficiency* (EE) $EE = E_{ANN}/E_{SNN}$ will remain unaffected with or without memory access consideration. Finally, to show the advantage of utilizing BackRes connections, we also compute the total number of unique parameters (i.e., total number of weights) in a network and calculate the compression ratio that BackRes blocks yield over conventional feedforward blocks of similar logical depth.

8. RESULTS

8.1. Impact of BackRes Connections

First, we show the impact of incorporating BackRes Connections for conversion based SNNs. **Table 2** compares the accuracy and total # parameters across different network topologies (described in **Table 3**) for ANN/SNN implementations on CIFAR10 data. For the sake of understanding, we provide the unrolled computation graph of networks with BackRes blocks and repeated computations in **Table 3**. For instance, VGG2x4 refers to a network which has two unique convolutional layers ($Conv1, Conv2$) where $Conv2$ receives a BackRes Connection from its output and is computed 4 times before processing

TABLE 2 | Accuracy and Total # parameters for ANN and corresponding converted SNN topologies (refer **Table 3**) for different latency T on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 250$)	SNN ($T = 2500$)	#Parameters
(Accuracy %)				
VGG7	88.74	85.88	88.56	1.2M (1x)
VGG2x4	86.14	81.99	86.23	1.09M (1.1x)
VGG3x2-v1	87.34	83.31	87.15	1.13M (1.06x)

TABLE 3 | CIFAR10 network topologies for Conversion training.

Model	Configuration	BackRes
VGG7	Input-Conv1(3,64,3 × 3/1)-Conv2(64,64,3 × 3/1)- -Conv3(64,64,3 × 3/1)-Conv4(64,64,3 × 3/1)- -Conv5(64,64,3 × 3/1)-Pool(2x2/2)-Pool(2 × 2/2)- -Pool(2 × 2/2)-FC1(2048,512)-FC2(512,10)	Not applicable
VGG2x4	Input-Conv1(3,64,3 × 3/1)-Conv2(64,64,3 × 3/1)- -Conv2(64,64,3 × 3/1)-Conv2(64,64,3 × 3/1)- -Conv2(64,64,3 × 3/1)-Pool(2 × 2/2)-Pool(2 × 2/2)- -Pool(2 × 2/2)-FC1(2048,512)-FC2(512,10)	[Conv2] repeated 4 times
VGG3x2-v1	Input-Conv1(3,64,3 × 3/1)-Conv2(64,64,3 × 3/1)- -Conv3(64,64,3 × 3/1)-Conv2(64,64,3 × 3/1)- -Conv3(64,64,3 × 3/1)-Pool(2 × 2/2)-Pool(2 × 2/2)- -Pool(2 × 2/2)-FC1(2048,512)-FC2(512,10)	[Conv2- Conv3] repeated 2 times

ConvN(I,O,k×k/s) denotes N^{th} convolutional layer with I input channels, O output channels, kernel of size $k \times k$ with stride s . Pool($p \times p/s_p$) denotes average pooling layer with pooling window size $p \times p$ and pooling stride s_p . FC(X, Y) denote a fully-connected layer with X input nodes and Y output nodes. Layers with BackRes connections and repeated computations have been highlighted in red.

the next layer as depicted in **Table 3**. Similarly, VGG3x2-v1 refers to a network with three unique convolutional layers ($Conv1, Conv2, Conv3$) with $Conv2, Conv3$ computation repeated two times in the order depicted in **Table 3**. Note, VGG2x4/VGG3x2-v1 achieve the same logical depth of a 7-unique layered (including fully connected layers) VGG7 network.

In **Table 2**, we observe that accuracy of ANNs with BackRes connections suffer minimal loss (upto $\sim 1 - 2\%$ loss) to that of the baseline ANN-VGG7 model. The corresponding converted SNNs with BackRes connections also yield near-accuracy. It is evident that SNNs with higher computation time or latency T yield better accuracy. While the improvement in total # parameters is minimal here, we observe a significant improvement in energy efficiency [$EE = \frac{E_{ANN}(1x)}{E_{SNN}}$ calculated using Equations (10), (11)] with BackRes additions as shown in **Figure 7**. Note, the EE of SNNs shown in **Figure 7** is plotted by taking the corresponding ANN topology as baseline (EE of VGG2x4 SNN is measured with respect to VGG2x4 ANN). The large efficiency gains observed can be attributed to the sparsity obtained with event-driven spike-based processing as well as the repeated computation achieved with BackRes connections. In fact, we find that net spiking activity for a given layer decreases over repeated computations (implying a “sparsifying effect”) with

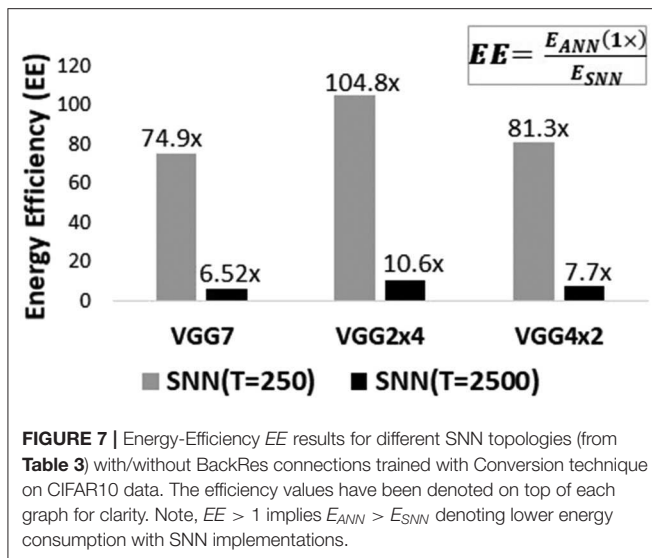


TABLE 4 | Accuracy, Total # parameters and Energy Efficiency EE for converted SNN topologies (refer **Table 5**) of latency $T = 2, 500$ and corresponding ANN on imagenet data.

Model	ANN ($T = 1$)	SNN ($T = 2500$)	#Parameters	$EE = \frac{E_{ANN}(1x)}{E_{SNN}}$
[Accuracy (Top-1/Top-5%)]				
VGG16	70.52/ 89.39	69.96/ 89.01	123.8M (1x)	1.975x
VGG11x2	69.72/ 88.56	68.57/ 87.66	116.1M (1.07x)	3.66x

each unrolling step (due to increasing threshold per unrolling, see section 4). Consequently, VGG2x4 with $n = 4$ repeated computation yields larger EE ($\sim 1.3\times$) than VGG3x2-v1 ($n = 2$).

Table 4 illustrates the Top-1/Top-5 accuracy, parameter compression ratio and EE benefits observed with BackRes connections on Imagenet dataset (for topologies shown in **Table 5**). Note, VGG11x2 (comprising of 11 unique convolutional or fully-connected layers) with BackRes connections and repeated computations achieves the same logical depth of 16 layers as that of VGG16. The accuracy loss in VGG11x2 (SNN) is minimal $\sim 1\%$ while yielding $\sim 2\times$ greater EE compared to VGG16 (SNN). We also find that for complex datasets like Imagenet, lower latency of $T = 250$ yields very low accuracy with or without BackRes computations.

Next, we evaluate the benefits of adding BackRes connections for SNNs trained with STDP. As discussed earlier, in STDP training, the convolutional layers of a network are trained layerwise with LIF neurons. Then, an ANN classifier is appended to the STDP trained layers, wherein, the ANN classifier is trained separately on the overall spiking activity collected at the SNN layers. **Table 6** shows the accuracy, # parameters and EE benefits of SNN topologies (listed in **Table 7**) with respect to corresponding ANN baselines. All ANN baselines are trained end-to-end with backpropagation and requires the

TABLE 5 | Imagenet network topologies for conversion training.

Model	Configuration	BackRes
VGG16	Input-Conv1(3,64,3x3/1)-Conv2(64,64,3x3/1)- -Pool(2 x 2/2)-Conv3(64,128,3 x 3/1)-Pool(2 x 2/2)-Conv4(128,256,3 x 3/1)- -Conv5(256,256,3 x 3/1)-Conv6(256,256,3 x 3/1)-Pool(2 x 2/2)-Conv7(256,512,3 x 3/1)- -Conv8(512,512,3 x 3/1)-Conv9(512,512,3 x 3/1)-Conv10(512,512,3 x 3/1)-Conv11(512,512,3 x 3/1)- -Conv12(512,512,3 x 3/1)-Conv13(512,512,3 x 3/1)-Pool(2 x 2/2)-Pool(2 x 2/2)- -FC1(25088,4096)-FC2(4096,1000)	Not applicable
VGG11x2	Input-Conv1(3,64,3 x 3/1)-Conv2(64,64,3 x 3/1)- -Pool(2 x 2/2)-Conv3(64,128,3 x 3/1)-Pool(2 x 2/2)-Conv4(128,256,3 x 3/1)-Conv5(256,256,3 x 3/1)- -Conv5(256,256,3 x 3/1)-Pool(2 x 2/2)-Conv6(256,512,3 x 3/1)-Conv7(512,512,3 x 3/1)- -Conv8(512,512,3 x 3/1)-Conv9(512,512,3 x 3/1)-Conv7(512,512,3 x 3/1)-Conv8(512,512,3 x 3/1)- -Conv9(512,512,3 x 3/1)-Pool(2 x 2/2)-Pool(2 x 2/2)- -FC1(25088,4096)-FC2(4096,1000)	[Conv5]& [Conv7- Conv8- Conv9] repeated 2 times

Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red.

TABLE 6 | Accuracy, Total # parameters and Energy Efficiency EE for STDP-trained SNN topologies (refer **Table 7**) of latency $T = 100$ and corresponding ANN on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 100$)	#Parameters	$EE_{Conv}/EE_{Full} = \frac{E_{ANN}(1x)}{E_{SNN}}$
(Accuracy%)				
ResNet2	78.26	61.02	18.9 M	1.64x/1.16x
ResNet3	80.11	51.1	28.37 M	1.81x/1.28x
ResNet2x2	79.39	63.21	28.35 M	10.56x/1.78x

EE_{Conv} considers the energy calculated only for the convolutional/pooling layers excluding the FC layers, EE_{Full} considers the total energy of the network including the FC layers.

entire CIFAR10 training dataset (50,000 labeled instances). On the other hand, all SNNs requires only 5,000 instances for training the Convolutional layers. Then, the fully-connected classifier (comprising of FC1, FC2 layers in **Table 7**) appended separately to the STDP-learned layers are trained on the entire CIFAR10 dataset.

From **Table 6**, we observe that SNN accuracy is considerably lower than corresponding ANN accuracy. This can be attributed to the limitation of STDP training to extract relevant features in an unsupervised manner. In fact, deepening the network from ResNet2 to ResNet3 causes a decline in accuracy corroborating the results of previous works (Srinivasan and Roy, 2019). However, adding BackRes connection in ResNet2x2 which achieves same logical depth as ResNet3 improves the accuracy of the network while yielding significant gains ($\sim 10\times$) in terms of EE . For EE , we show the gains considering the full network EE_{Full}

TABLE 7 | CIFAR10 network topologies for STDP training methodology.

Model	Configuration	BackRes	Skip
ResNet2	Input-Conv1(3,36,3 × 3/1)-Conv2(36,36,3 × 3/1)- -Pool(2 × 2/2)-FC1(18432,1024)-FC2(1024,10)	Not applicable	Input-to-Conv2, Conv1-to-FC1
ResNet3	Input-Conv1(3,36,3 × 3/1)- -Conv2(36,36,3 × 3/1)- -Conv3(36,36,3 × 3/1)-Pool(2 × 2/2)- -FC1(27648,1024)-FC2(1024,10)	Not applicable	Input-to-Conv2, Conv1-to-FC1, Conv2-to-FC1
ResNet2x2	Input-Conv1(3,36,3 × 3/1)- -Conv2(36,36,3 × 3/1)- -Conv2(36,36,3 × 3/1)-Pool(2 × 2/2)- -FC1(18432,1024)-FC2(1024,10)	[Conv2] repeated 2 times	Input-to-Conv2, Conv1-to-FC1

Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red. Forward Residual or Skip connections between layers of a network are denoted in blue.

TABLE 8 | Accuracy, Total, # parameters, and Energy Efficiency EE for AGD trained SNN topologies (refer **Table 9**) of latency $T = 25, 50$, and corresponding ANN on CIFAR10 data.

Model	ANN ($T = 1$)	SNN ($T = 25$)	SNN ($T = 50$)	#Parameters	$EE = \frac{E_{ANN}(T \times)}{E_{SNN}(T=25)}$
(Accuracy%)					
VGG5	75.86	71.92	72.77	2.21M	14.75x
VGG3x2-v2	74.99	71.07	71.97	2.18M	16.2x
VGG7	72.26	—	—	2.3M	—
VGG3x4	69.52	74.23	75.01	2.19M	26.44x

TABLE 9 | CIFAR10 network topologies for AGD training methodology.

Model	Configuration	BackRes
VGG5	Input-Conv1(3,64,3x3/1)-Conv2(64,64,3x3/1)- -Pool(2x2/2)-Conv3(3,64,3x3/1)-Conv4(64,64,3x3/1)- -Pool(2x2/2)-FC1(4096,512)-FC2(512,10)	Not applicable
VGG3x2-v2	Input-Conv1(3,64,3x3/1)-Conv2(64,64,3x3/1)- -Pool(2x2/2)-Conv3(3,64,3x3/1)-Conv3(64,64,3x3/1)- -Pool(2x2/2)-FC1(4096,512)-FC2(512,10)	[Conv3] repeated 2 times
VGG7	Input-Conv1(3,64,3x3/1)-Conv2(64,64,3x3/1)- -Pool(2x2/2)-Conv3(3,64,3x3/1)-Conv4(64,64,3x3/1)- -Conv5(3,64,3x3/1)-Conv6(64,64,3x3/1)- -Pool(2x2/2)-FC1(4096,512)-FC2(512,10)	Not applicable
VGG3x4	Input-Conv1(3,64,3x3/1)-Conv2(64,64,3x3/1)- -Pool(2x2/2)-Conv3(3,64,3x3/1)-Conv3(64,64,3x3/1)- -Conv3(3,64,3x3/1)-Conv3(64,64,3x3/1)- -Pool(2x2/2)-FC1(4096,512)-FC2(512,10)	[Conv3] repeated 4 times

Notations are same as that of **Table 3**. Layers with BackRes connections and repeated computations have been highlighted in red.

(including spiking convolutional and ReLU FC layers), as well as, the gain considering only the spiking convolutional layers EE_{Conv} . The spiking layers on account of event-driven sparse computing exhibit higher efficiency than the full network (i.e., $EE_{Conv} >$

EE_{Full}). Interestingly, ResNet2x2 yields $\sim 10\times$ higher efficiency at the spiking layers which further supports the fact that BackRes connections have a “sparsifying” effect on the intrinsic spiking dynamics of the network. This result establishes the advantage of BackRes connection in enabling scalability of STDP-based SNN training methodologies toward larger logical depth while yielding both accuracy and efficiency improvements.

For AGD training, BackRes additions yield both accuracy and scalability related benefits. **Table 8** shows the accuracy, # parameters and EE benefits of SNN topologies (listed in **Table 9**) for different latency $T = 25, 50$ with respect to corresponding ANN baselines. Similar to Conversion/STDP results, end-to-end AGD training with spiking statistics (using surrogate gradient descent) for VGG5 and VGG3x2-v2 of equivalent logical depth as VGG5 yields minimal accuracy loss ($\sim 2 - 3\%$) and large EE gains ($\sim 15\times$) in comparison to corresponding ANNs. However, for a VGG7 network with 7-layered depth, AGD fails to train an SNN end-to-end due to vanishing forward spike-propagation. Interestingly, a VGG3x4 network with similar logical depth of 7-layers as VGG7 and repeated computations not only trains well with AGD, but also yields higher accuracy than both VGG7/VGG3x4 ANN baselines. This implies that LIF neurons with spiking statistics have the potential of yielding more diversified computation profile with BackRes unrolling than ReLU neurons. In addition to accuracy and scalability benefits, SNNs with BackRes connections yield high EE benefits as shown in **Table 8** (due to the inherent “sparsifying” effect) that point to their suitability for low-power hardware implementations.

8.2. Impact of Stochmax

Stochmax is essentially a classification-performance improvement technique that can result in improved latency benefits. First, we show the impact of incorporating stochmax classifier for SNNs trained with AGD. **Table 10** compares the accuracy of small VGG3 SNN trained with AGD for different latency T . Here, the FC2 layer of VGG3 topology is implemented as a softmax or stochmax classifier. We observe a consistent improvement in accuracy for stochmax implementations. In **Table 11**, we show the accuracy results for SNNs of

TABLE 10 | Accuracy for AGD trained SNN of VGG3 topology (refer to last row) for different latency $T = 5, 10, 25$ on CIFAR10 data.

Model	$T = 5$	$T = 10$	$T = 25$
VGG3 (StochMax)	50.4	65.24	70.2
VGG3 (SoftMax)	49.1	64.44	67.1
VGG3 (Topology)	Input-Conv1(3,64,3x3/1)-Pool(2x2/2)-Conv2(64,64,3x3/1)-Pool(2x2/2)-FC1(4096,512)-FC2(512,10)		

TABLE 11 | Accuracy and EE benefits for AGD trained SNN with stochmax classifier on VGG5/VGG3x2-v2 topology (refer to **Table 9**) for different latency $T = 25, 50$ on CIFAR10 data.

Model	$T = 25$	$T = 50$	$EE = \frac{E_{ANN}(1x)}{E_{SNN}}$	$EE = \frac{E_{SNN}(softmax)}{E_{SNN}(stochmax)}$
	(Accuracy%)		(for $T = 25$)	
VGG5	75.26	75.92	23.83x	1.62x
VGG3x2-v2	72.62	73.17	31.88x	1.97x

TABLE 12 | Accuracy and EE benefits for STDP trained SNN with ConvNN classifier (**Table 13**) appended to ResNet2, ResNet2x2, ResNet3 topology (refer to **Table 7**) on CIFAR10 data.

Model	ANN $T = 1$	SNN $T = 100$	$EE_{Conv}/EE_{Full} = \frac{E_{ANN}(1x)}{E_{SNN}}$
	(Accuracy%)		
ResNet2	83.5	77.92	1.64x/1.08x
ResNet3	79.85	76.52	1.81x/1.69x
ResNet2x2	83.2	80.1	10.56x/2.14x

EE_{Conv} considers the energy calculated only for the convolutional/pooling layers excluding the FC layers, EE_{Full} considers the total energy of the network including the FC layers.

VGG5/VGG3x2-v2 topology with stochmax classifiers. It is evident that stochmax improves the performance by $\sim 3 - 4\%$ as compared to softmax implementations in **Table 8**. In addition to accuracy, we also observe a larger gain in energy-efficiency with stochmax implementations. We find that conducting end-to-end AGD training with stochmax loss leads to sparser spiking activity across different layers of a network as compared to softmax. We believe this might be responsible for the efficiency gains. Further theoretical investigation is required to understand the role of loss optimization in a temporal processing landscape toward decreasing the spiking activity without affecting the gradient values. **Tables 10, 11** results suggest stochmax as a viable technique for practical applications where we need to obtain higher accuracy and energy benefits with constrained latency or processing time.

Inclusion of stochmax classifier in SNNs trained with conversion/STDP training results in a slight improvement in accuracy $\sim 1 - 2\%$ for CIFAR10 data (for VGG7/ResNet3 topologies from **Tables 2, 6**), respectively. Since stochmax is dissociated from the training process in both STDP/conversion,

TABLE 13 | ConvNN classifier network topologies for STDP training methodology.

Model	Configuration
ConvNN ResNet2, ResNet2x2	Input-Conv1(72,72,3x3/1)-Conv2(72,72,3x3/1)-Pool(2x2/2)-Conv3(72,144,3x3/1)-Conv4(144,144,3x3/1)-Pool(2x2/2)-FC1(2304,1024)-FC2(1024,10)
ConvNN ResNet3	Input-Conv1(108,108,3x3/1)-Conv2(108,108,3x3/1)-Pool(2x2/2)-Conv3(108,216,3x3/1)-Conv4(216,216,3x3/1)-Pool(2x2/2)-FC1(3456,1024)-FC2(1024,10)

Notations are same as that of **Table 3**.

TABLE 14 | Accuracy, Total, # parameters, and Energy Efficiency EE for AGD trained SNN topologies (refer **Table 15**) with hybrid ReLU/LIF neurons of latency $T = 25$ and corresponding ANN on CIFAR10 data.

Model	ANN $T = 1$	SNN $T = 25$	#Parameters	$EE = \frac{E_{ANN}(1x)}{E_{SNN}}$
	(Accuracy%)			
VGG9	83.33	84.98	5.96M	3.98x
VGG8x2	83.49	84.26	5.37M	4.1x

TABLE 15 | Network topologies for AGD training methodology with hybrid layers and stochmax classifier at the end.

Model	Configuration	BackRes
VGG9	Input-Conv1(3,64,3x3/1)-ReLU-Conv2(64,64,3x3/1)-ReLU-Pool(2x2/2)-Conv3(64,128,3x3/1)-LIF-Conv4(128,128,3x3/1)-LIF-Pool(2x2/2)-Conv5(128,256,3x3/1)-LIF-Conv6(256,256,3x3/1)-LIF-Conv7(256,256,3x3/1)-LIF-Pool(2x2/2)-FC1(4096,1024)-LIF-FC2(1024,10)	Not applicable
VGG8x2	Input-Conv1(3,64,3x3/1)-ReLU-Conv2(64,64,3x3/1)-ReLU-Pool(2x2/2)-Conv3(64,128,3x3/1)-LIF-Conv4(128,128,3x3/1)-LIF-Pool(2x2/2)-Conv5(128,256,3x3/1)-LIF-Conv6(256,256,3x3/1)-LIF-Conv6(256,256,3x3/1)-LIF-Pool(2x2/2)-FC1(4096,1024)-LIF-FC2(1024,10)	

Notations are same as that of **Table 3**.

the latency and energy efficiency results are not affected. Note, all results shown in **Tables 2–8** use softmax classifier.

8.3. Impact of Hybridization

Except for Conversion, both STDP and AGD training techniques fail to yield high accuracy for deeper network implementations. While BackRes connections and Stochmax classifiers improve

the accuracy, an SNN still lags behind its corresponding ANN in terms of performance. To improve the accuracy, we employ hybridization with partially ReLU and partially LIF neurons for SNN implementations.

For STDP, we strengthen the classifier that is appended to the STDP trained convolutional layers to get better accuracy. Essentially, we replace the fully-connected layers *FC1*, *FC2* of the topologies in **Table 7** with a larger convolutional network *ConvNN* (*ConvNN* topology description is given in **Table 13**). **Table 12** shows the accuracy, *EE* results for the STDP trained ResNet topologies appended now with corresponding *ConvNN* and compared to a similar ANN baseline (say, ResNet2 ANN corresponds to an ANN with ResNet2 topology with FC layers replaced by *ConvNN* classifier from **Table 13**). Strengthening the classifier hierarchy now results in higher accuracies ($\sim 75\%$) comparable to the ANN performance of **Table 6**, while still lagging behind the ANN baseline of similar topology. However, the accuracy loss between ANN and SNN in this case reduces quite significantly ($> 20\%$ loss in **Table 6** to $\sim 3\%$ loss in **Table 12**). Similar to **Table 6**, for *EE*, the gains considering only spiking layers are greater than that of the full network.

For AGD, as discussed in section 6, we hybridize our network with initial layers comprising of ReLU and latter layers of LIF neurons and perform end-to-end gradient descent. **Table 14** shows the accuracy and *EE* gain results for a VGG9, VGG8x2 model (topology description in **Table 15**) with BackRes connection trained using hybridization for CIFAR10 data. Note, only the first two convolutional layers *Conv1*, *Conv2* use ReLU activation, while the remaining layers use LIF functionality. In addition, we use a softmax classifier at the end instead of softmax to get better accuracy. Earlier, we saw that a 7-layered network could not be trained with AGD (see **Table 8**). Inclusion of ReLU layers now allows a deep 9-layered network to be trained end-to-end while yielding considerable energy-efficiency gain with slightly improved accuracy ($\sim 1\%$ improvement in accuracy in SNN) in comparison to a corresponding ANN baseline (note, ANN baseline has ReLU activation in all layers). To have fair comparison between ANN and SNN, ANN baselines are trained without any batch normalization or other regularization techniques. Including batch normalization and dropout in ANN training yields $\sim 86\%$ accuracy that is still fairly close to $\sim 85\%$ accuracy obtained with the SNN implementations. To calculate *EE* gains in hybrid SNN implementations, we consider MAC energy for ReLU layers (*Conv1*, *Conv2* in **Table 14**) and AC energy for remaining LIF layers (*Conv3* – *Conv7*(6) in **Table 14**). VGG8x2 achieves equivalent logical depth as VGG9. Similar to earlier results, VGG8x2 yields slightly higher benefit than VGG9 on account of the “sparsifying” effect induced by BackRes computations.

Table 16 shows the results of a VGG13 model (topology description in **Table 17**) trained with hybrid ReLU/LIF neuron layers on Imagenet dataset learn with end-to-end gradient descent. Interestingly, for Imagenet data, we had to use ReLU neuronal activations both in the beginning as well as at the end as shown in **Table 17**. After some trial-and-error analysis, we found that training with more LIF neuronal layers for a complex dataset like Imagenet did not yield good performance.

TABLE 16 | Accuracy and Energy Efficiency *EE* for AGD trained SNN topologies (refer **Table 17**) with hybrid ReLU/LIF neurons of latency $T = 10$ and corresponding ANN on Imagenet data.

Model	ANN $T = 1$	SNN $T = 10$	$EE = \frac{E_{ANN}(1 \times)}{E_{SNN}}$
	(Accuracy%)		
VGG13	Top-1 69.9 Top-5 89.9	Top-1 67.6 Top-5 88.23	1.31x

TABLE 17 | Network topologies for AGD training methodology with hybrid layers and softmax classifier at the end for imagenet dataset.

Model	Configuration	BackRes
VGG13	Input-Conv1(3,64,3x3/1)-ReLU- -Conv2(64,64,3x3/1)-ReLU-Pool(2x2/2)- -Conv3(64,128,3x3/1)-ReLU- -Conv4(128,128,3x3/1)-ReLU-Pool(2x2/2)- -Conv5(128,256,3x3/1)-ReLU- -Conv6(256,256,3x3/1)-ReLU-Pool(2x2/2)- -Conv7(256,512,3x3/1)-LIF- -Conv8(512,512,3x3/1)-LIF-Pool(2x2/2)- -Conv9(512,512,3x3/1)-ReLU- -Conv10(512,512,3x3/1)-ReLU-Pool(2x2/2)- -FC1(25088,4096)-ReLU-FC2(4096,4096) -FC3(4096,1000)	Not applicable

Notations are same as that of **Table 3**.

In case of a VGG13 network, converting the middle two layers into spiking LIF neurons yielded iso-accuracy as that of a fully-ReLU activation based ANN. Even with a minor portion of the network offering sparse neuronal spiking activity, we still observe $1.3\times$ improvement in *EE* with our hybrid model over the standard ANN. It is also worth mentioning that the spiking LIF neurons of the hybrid VGG13 network have a lower processing latency of $T = 10$. We believe that using ReLU activations in majority of the VGG13 network enabled us to process the spiking layers at lower latency. We can expect higher *EE* gains by adding suitable backward residual connections in the spiking layers to compensate for depth. It is evident that hybridization incurs a natural tradeoff between number of spiking/ReLU layers, processing latency, accuracy and energy-efficiency. Our analysis shows that hybridization can enable end-to-end backpropagation training for large-scale networks on complex datasets while yielding efficiency gains. Further investigation is required to evaluate the benefits of hybridization in large-scale setting by varying the tradeoff parameters.

9. DISCUSSION AND CONCLUSION

With the advent of Internet of Things (IoT) and the necessity to embed intelligence in devices that surround us (such, smart phones, health trackers), there is a need for novel computing solutions that offer energy benefits

while yielding competitive performance. In this regard, SNNs driven by sparse event-driven processing hold promise for efficient hardware implementation of real-world applications. However, training SNNs for large-scale tasks still remains a challenge. In this work, we outlined the limitation of the three widely used SNN training methodologies (Conversion, AGD training and STDP), in terms of, *scalability*, *latency*, and *accuracy*, and proposed novel solutions to overcome them.

We propose using backward residual (or BackRes) connections to achieve logically deep SNNs with shared network computations and features that can approach the accuracy of fully-deep SNNs. We show that all three training methods benefit from the BackRes connection inclusion in the network configuration, especially, gaining in terms of energy-efficiency ($\sim 10 \times -100 \times$) while yielding iso-accuracy with that of an ANN of similar configuration. We also find that BackRes connections induce a *sparsifying effect* on overall network activity of an SNN, thereby, expending lower energy ($\sim 1.8 - 3.5 \times$ lower) than an equivalent depth full-layered SNN. In summary, BackRes connections address the scalability limitations of an SNN that arise due to depth incompatibility and *vanishing spike-propagation* of different training techniques.

We propose using stochastic softmax (or stochmax) to improve the prediction capability of an SNN, specifically, for AGD training method that uses end-to-end spike-based backpropagation. We find a significant improvement in accuracy ($\sim 2 - 3\%$) with stochmax inclusion even for lower latency or processing time period. Further, stochmax loss based backpropagation results in lower spiking activity than the conventional softmax loss. Combining the advantages of lower latency and sparser activity, we get higher energy-efficiency improvements ($\sim 1.6 - 2 \times$) with stochmax SNNs as compared to softmax SNNs. Conversion/STDP training do not benefit in terms of efficiency and latency from stochmax inclusion since the training in these cases are performed fully/partially with ANN computations.

The third technique we propose is using a hybrid architecture with partly-ReLU-and-partly-LIF computations in order to improve the accuracy obtained with STDP/AGD training methods. We find that hybridization leads to improved accuracy at lower latency for AGD/STDP methods, even circumventing the inadequacy of training very deep networks. The accuracies observed for CIFAR10 ($\sim 80\%/85\%$) with STDP/AGD on hybrid SNN architectures are in fact comparable/better than ANNs of similar configuration. We would like to note that hybridization also offers significant energy-efficiency improvement ($\sim 4 \times$) over a fully ReLU-based ANN. In fact, using hybridization, we trained a deep VGG13 model on Imagenet data and obtained iso-accuracy as that of its ANN counterpart with reasonable energy-efficiency gains. There are interesting possibilities of performing distributed edge-cloud intelligence with such hybrid SNN-ANN architecture where, SNN layers can be implemented on resource-constrained edge devices and ANN layers on the cloud.

9.1. Latency-Based Coding vs. Rate Coding

Across all SNN implementations in this work, we used rate coding to convert pixel data of images into spike trains. However, it is known that rate coding does not allow the network to use spike-times precisely which can, in turn, enable an SNN to encode more information or process information rapidly. Supervised learning based SNNs using latency-based coding scheme is a good way to decrease the energy consumption, compared to the rate-coding method (Mostafa, 2017; Comsa et al., 2019; Kheradpisheh and Masquelier, 2019; Zhou et al., 2019). In latency-based coding, pixel intensity is represented by the ascending order of incoming spikes, wherein higher intensity fires an earlier spike and vice-versa. As a result, more salient information about a feature is encoded as an earlier spike in the corresponding neuron leading to overall sparser activity in an SNN. Furthermore, the inference latency (or overall time steps required to process an input) can drastically decrease to few 10 time steps with appropriate learning methods instead of the usual 50–100 time steps incurred in rate coding schemes for AGD training (Mostafa, 2017; Comsa et al., 2019; Kheradpisheh and Masquelier, 2019; Roy et al., 2019; Zhou et al., 2019).

Mostafa (2017) proposed a direct training based method via backpropagation error and the networks have achieved very high accuracy on MNIST compared to the other unsupervised learning or conversion based SNNs. Nevertheless, the networks proposed by Mostafa et al. have not been applied to the more complicated dataset, such as CIFAR10. The reasons are convolutional layers are not included and the preprocessing method is not general. In Zhou et al. (2019), the authors incorporate convolutional layers into the SNNs proposed by Mostafa et al. In addition, they propose a new way to preprocess the input data and develop a new kernel operation process without using ReLU activation. With these new additions, Zhou et al. present the best results obtained so far on a purely temporal based backpropagation learning scheme for CIFAR10 (80.5%). In addition, other recent works (Comsa et al., 2019; Kheradpisheh and Masquelier, 2019) have also shown impressive and promising results on the use of first-to-spike latency coding in realizing the energy-efficiency and inference latency benefits with SNN implementations. However, the framework (including the network architecture and learning rule used to perform spike-based backpropagation) is very different in each work. The inconsistencies in the implementations as well as the algorithmic details are a slight drawback in arriving at a uniform testbed implementation with latency-coded techniques.

Furthermore, while rate-coded schemes (specifically, with conversion training methodology) are approaching ANN-like competitive performance on a host of datasets (including, Imagenet), latency-based coding still suffer from accuracy limitations. However, we believe that latency based coding can bring out the true advantages of SNNs (both for competitive accuracy and higher efficiency gains) compared to ANNs. One advantage of latency-based backpropagation and using AGD, is that they can make use of temporal coding, so they can actually outperform an ANN on the same architecture. We see a hint of this in our AGD trained SNN implementation in **Table 8**

(SNN implementation of VGG3x4 model has higher accuracy (75%) than corresponding ANN (69.52%)). In the future, we will explore the advantages of incorporating our proposed backward residual connection, stochmax, and hybrid training schemes on such latency coded networks and study their impact on the scalability, latency, and accuracy limitations observed in SNNs.

9.2. Connection Between Binary ANNs and SNNs

Binary ANNs (Courbariaux et al., 2016; Rastegari et al., 2016) are extreme quantized form of neural networks that have neuronal activations and weights represented as binary values. Thus, binary ANNs have been shown to yield considerable memory compression and energy efficiency improvements over conventional full precisions ANNs. Thus, an obvious question one can ask is, “How SNNs stand against binary ANNs?” In a recent work by Lu and Sengupta (2020), the authors show an interesting connection between binarized ANNs and SNNs with a conversion methodology. Basically, the authors argue that ANN-SNN conversion provides a mathematical formulation for expressing multi-bit precision of ANN activations as binary values over time (in the context of an SNN). The authors achieve binary SNN models that yield near full-precision accuracies on large-scale image recognition datasets, while utilizing similar hardware backbone of binary neural network catered “In-Memory” computing platforms. The fact that binarized ANNs have also simplified accumulate operation (instead of multiply and accumulate) similar to that of an SNN can result in lower energy savings that one would expect in comparison to a full-precision ANN. In Lu et al., the authors show that a binary SNN obtained by converting a constrained ANN on CIFAR100 dataset has $\sim 4\times$ higher computational cost (measured in terms of number of multiply-accumulate logic operations performed) than an XNOR-net (Rastegari et al., 2016) of similar architecture. On the other hand, the binarized SNN yields 20% higher accuracy (nearly similar to that of the full-precision ANN model) than the XNOR model. It is well-known that training Binary ANNs (Courbariaux et al., 2016), XNOR-nets (Rastegari et al., 2016) from scratch can be prohibitive in terms of training convergence (due to the fact that the neuronal activations are constrained to +1 or -1 or 0). In that regard, if we would like to deploy binarized SNNs, using a strategy similar to Lu and Sengupta (2020) will be useful. While we will lose in terms of efficiency, we will tradeoff the slight decremented efficiency with a large increase in accuracy. Furthermore, in our paper’s context, we conjecture that since the SNN training convergence improved

in some cases with modifications like backward residual training and stochmax, training a Binary ANN with such modifications can potentially give accuracy benefits. In the future, we will investigate how binary neural networks can be used to generate low-precision SNNs through conversion, STDP, AGD training. In fact, training a hybrid ANN-SNN model, where the ANN comprises of binarized weights and activations, can potentially give us higher order efficiency gains that requires further investigation.

Finally, SNNs are a prime candidate today toward enabling low-powered ubiquitous intelligence. In this paper, we show the benefit of using good practices while configuring spiking networks to overcome their inherent training limitations, while, gaining in terms of energy-efficiency, latency, and accuracy for image recognition applications. In the future, we will investigate the extension of the proposed methods for training recurrent models for natural language or video processing tasks. Further, conducting reinforcement learning with the above proposed techniques to analyze the advantages that SNNs offer is another possible future work direction.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/supplementary material.

AUTHOR CONTRIBUTIONS

PP developed the main concepts, performed the simulations, and wrote the paper. SA helped in performing the simulations. All authors assisted in the writing of the paper.

FUNDING

This work was supported in part by C-BRIC, Center for Brain Inspired Computing, a JUMP center sponsored by DARPA and SRC, by the Semiconductor Research Corporation, the National Science Foundation (Grant# 1947826), Amazon Research Award, Intel Corporation, the Vannevar Bush Faculty Fellowship and the U.K. Ministry of Defense under Agreement Number W911NF-16-3-0001. The authors declare that this study received funding from Intel Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication. This manuscript has been released as a pre-print at arXiv.org (arXiv:1910.13931) (Panda et al., 2019).

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.
- 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* (San Francisco, CA: ACM), 27. doi: 10.1145/3061639.3062311
- 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* (Montreal, QC), 787–797.

- 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
- Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. (2019). Temporal coding in spiking neural networks with alpha synaptic function. *arXiv preprint arXiv:1907.13223*. doi: 10.1109/ICASSP40776.2020.9053856
- Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL), 248–255. doi: 10.1109/CVPR.2009.5206848
- 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), 1–8. doi: 10.1109/IJCNN.2015.7280696
- 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* (Montreal, QC), 1135–1143.
- 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. doi: 10.1109/CVPR.2016.90
- Hunsberger, E., and Eliasmith, C. (2015). Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*.
- Indiveri, G., Corradi, F., and Qiao, N. (2015). "Neuromorphic architectures for spiking deep neural networks," in *2015 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC), 4–12. doi: 10.1109/IEDM.2015.7409623
- Indiveri, G., and Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. *Front. Neurosci.* 5:118. doi: 10.3389/fnins.2011.00118
- 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
- Kheradpisheh, S. R., and Masquelier, T. (2019). S4NN: temporal backpropagation for spiking neural networks with one spike per neuron. *arXiv preprint arXiv:1910.09495*. doi: 10.1142/S0129065720500276
- Kubilius, J., Schrimpf, M., Nayeibi, A., Bear, D., Yamins, D. L. K., and DiCarlo, J. J. (2018). Cornet: modeling the neural mechanisms of core object recognition. *bioRxiv [Preprint]*. doi: 10.1101/408385
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521:436. doi: 10.1038/nature14539
- LeCun, Y., Cortes, C., and Burges, C. (2010). *MNIST Handwritten Digit Database*. AT&T Labs [Online]. Available online at: <http://yann.lecun.com/exdb/mnist>
- 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., Sarwar, S. S., and Roy, K. (2019). Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *arXiv preprint arXiv:1903.06379*. doi: 10.3389/fnins.2020.00119
- Lee, C., Srinivasan, G., Panda, P., and Roy, K. (2018b). Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity. *IEEE Trans. Cogn. Dev. Syst.* 11, 384–394. doi: 10.1109/TCDS.2018.2833071
- Lee, H. B., Lee, J., Kim, S., Yang, E., and Hwang, S. J. (2018). "Dropmax: adaptive variational softmax," in *Advances in Neural Information Processing Systems* (Montreal, QC), 919–929.
- 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
- Linares-Barranco, B., Serrano-Gotarredona, T., Camu nas-Mesa, L. A., Perez-Carrasco, J. A., Zamarre no-Ramos, C., and Masquelier, T. (2011). On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Front. Neurosci.* 5:26. doi: 10.3389/fnins.2011.00026
- Lu, S., and Sengupta, A. (2020). Exploring the connection between binary and spiking neural networks. *arXiv [Preprint]*. arXiv:2002.10064.
- 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
- Mostafa, H. (2017). 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
- Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595
- 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
- Panda, P., Aketi, A., and Roy, K. (2019). Towards Scalable, Efficient and Accurate Deep Spiking Neural Networks with Backward Residual Connections, Stochastic Softmax and Hybridization. *arXiv [Preprint]*. arXiv:1910.13931.
- Panda, P., Allred, J. M., Ramanathan, S., and Roy, K. (2017). 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
- 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. doi: 10.1109/IJCNN.2016.7727212
- 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., Zamarre no-Ramos, C., Serrano-Gotarredona, T., and Linares-Barranco, B. (2010). "On neuromorphic spiking architectures for asynchronous STDP memristive systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris), 1659–1662. doi: 10.1109/ISCAS.2010.5537484
- Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774
- 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. doi: 10.1007/978-3-319-46493-0_32
- 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
- Sengupta, A., Banerjee, A., and Roy, K. (2016). 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., and Roy, K. (2017). Encoding neural and synaptic functionalities in electron spin: a pathway to efficient neuromorphic computing. *Appl. Phys. Rev.* 4:041105. doi: 10.1063/1.5012763
- 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
- Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J., and Aimone, J. B. (2019). Training deep neural networks for binary communication with the whetstone method. *Nat. Mach. Intell.* 1, 86–94. doi: 10.1038/s42256-018-0015-y
- Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv [Preprint]*. arXiv:1409.1556

- 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. *ACM J. Emerg. Technol. Comput. Syst.* 14:44. doi: 10.1145/3266229
- 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
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Boston, MA), 1–9. doi: 10.1109/CVPR.2015.7298594
- van de Burgt, Y., Lubberman, E., Fuller, E. J., Keene, S. T., Faria, G. C., Agarwal, S., et al. (2017). A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nat. Mater.* 16:414. doi: 10.1038/nmat4856
- Voelker, A. R., Rasmussen, D., and Eliasmith, C. (2020). A spike in performance: training hybrid-spiking neural networks with quantized activation functions. *arXiv [Preprint]*. arXiv:2002.03553.
- Wang, Z., Joshi, S., Savel'ev, S. E., Jiang, H., Midya, R., Lin, P., et al. (2017). Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nat. Mater.* 16:101. doi: 10.1038/nmat4756
- Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560. doi: 10.1109/5.58337
- Zhou, S., Chen, Y., Ye, Q., and Li, J. (2019). Direct training based spiking convolutional neural networks for object recognition. *arXiv [Preprint]*. arXiv:1909.10837

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.

Copyright © 2020 Panda, Aketi and Roy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design

Maryam Parsa^{1,2*}, John P. Mitchell², Catherine D. Schuman², Robert M. Patton², Thomas E. Potok² and Kaushik Roy¹

¹ Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States, ² Computational Data Analytics, Oak Ridge National Laboratory, Oak Ridge, IN, United States

OPEN ACCESS

Edited by:

Hesham Mostafa,
Intel, United States

Reviewed by:

Shimeng Yu,
Georgia Institute of Technology,
United States
Bernhard Vogginger,
Technische Universität Dresden,
Germany

*Correspondence:

Maryam Parsa
mparsa@purdue.edu

Specialty section:

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

Received: 15 January 2020

Accepted: 02 June 2020

Published: 21 July 2020

Citation:

Parsa M, Mitchell JP, Schuman CD, Patton RM, Potok TE and Roy K (2020) Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design. *Front. Neurosci.* 14:667. doi: 10.3389/fnins.2020.00667

In resource-constrained environments, such as low-power edge devices and smart sensors, deploying a fast, compact, and accurate intelligent system with minimum energy is indispensable. Embedding intelligence can be achieved using neural networks on neuromorphic hardware. Designing such networks would require determining several inherent hyperparameters. A key challenge is to find the optimum set of hyperparameters that might belong to the input/output encoding modules, the neural network itself, the application, or the underlying hardware. In this work, we present a hierarchical pseudo agent-based multi-objective Bayesian hyperparameter optimization framework (both software and hardware) that not only maximizes the performance of the network, but also minimizes the energy and area requirements of the corresponding neuromorphic hardware. We validate performance of our approach (in terms of accuracy and computation speed) on several control and classification applications on digital and mixed-signal (memristor-based) neural accelerators. We show that the optimum set of hyperparameters might drastically improve the performance of one application (i.e., 52–71% for Pole-Balance), while having minimum effect on another (i.e., 50–53% for RoboNav). In addition, we demonstrate resiliency of different input/output encoding, training neural network, or the underlying accelerator modules in a neuromorphic system to the changes of the hyperparameters.

Keywords: multi-objective hyperparameter optimization, Bayesian optimization, neuromorphic computing, spiking neural networks, accurate and energy-efficient machine learning

1. INTRODUCTION

Neuromorphic systems promise a novel alternative to the standard von-Neumann architectures that are computationally expensive for analyzing big data, and are not efficient for learning and inference. This novel generation of computing aims at “mimicking” the human brain based on deploying neural networks on event-driven hardware architectures. A key bottleneck in designing such brain-inspired architectures is the complexity of co-optimizing the algorithm’s speed and accuracy along with the hardware’s performance and energy efficiency. This complexity stems from numerous intrinsic hyperparameters in both software and hardware that need to be optimized for an optimum design.

In this work we propose a novel optimization framework built upon agent-based modeling and hierarchical Bayesian optimization techniques to obtain the optimum set of hyperparameters for neuromorphic system design. Bayesian optimization is a powerful tool for finding the optimal point of objective functions that are unknown and expensive to evaluate (Shahriari et al., 2015). However, for problems with more than one objective function Bayesian-only techniques are mathematically complex, and suffer from high dimensionality limitations in parameter-heavy models (Dai et al., 2019). Other approaches such as Neural Architecture Search (NAS, Zoph et al., 2018) also require massive computational resources. These factors were the driving forces to search for alternative algorithms to find the optimal set of hyperparameters.

Our proposed approach, Hierarchical Pseudo Agent-based Bayesian Optimization (Hierarchical-PABO), is built upon using a supervisor agent correlating the results of isolated Bayesian estimations for each of the objective functions. The agent creates an extra set of Bayesian estimator focusing only on finding the Pareto frontier. The hierarchy of Bayesian optimizers enables predicting the Pareto frontier for complex problems regardless of the number of objective functions. In comparison with our previous works in (Parsa et al., 2019a,b), H-PABO is a general framework that covers both PABO (Parsa et al., 2019a) and single-objective Bayesian optimization (Parsa et al., 2019b) under its umbrella. In Parsa et al. (2019a), we introduced PABO, which was the initial phase toward designing Hierarchical PABO. PABO has no hierarchy of Bayesian estimators, and the supervisor agent decides the search direction in favor of the Pareto region, without any Bayesian estimator. By turning off the extra set of Bayesian estimators that are used to predict the Pareto frontier, H-PABO reduces to PABO. In Parsa et al. (2019b), we used a single objective hyperparameter Bayesian optimization to optimize performance of spiking neuromorphic systems in terms of neural network's accuracy. We showed how critical it is to use hyperparameter optimization techniques for designing any neuromorphic computing framework and how Bayesian approaches can help in this regard. H-PABO reduces to a single objective hyperparameter optimization problems when the number of objectives functions are fixed to one.

We tested Hierarchical-PABO on both artificial neural networks and spiking neural networks. For artificial neural networks, we validated our approach using AlexNet (Krizhevsky et al., 2012) and VGG19 (Simonyan and Zisserman, 2014) on a Programmable Ultra-Efficient Memristor-based Accelerator (PUMA, Ankit et al., 2019). For spiking neuromorphic systems, we considered several control and classification tasks such as the canonical pole balancing (Gomez et al., 2006), autonomous robotic navigation (Mitchell et al., 2017), satellite radio signal classification (Reynolds et al., 2018), and Iris dataset classification (Dua and Graff, 2017) on both digital and mixed-signal memristor-based accelerators as the underlying hardware (Chakma et al., 2017; Mitchell et al., 2018; Plank et al., 2018). Hierarchical-PABO predicts the Pareto frontier for a three-objective (network performance, the accelerator's energy efficiency, and area) optimization with relatively few evaluations. Compared to the state-of-the-art methods, our

framework is faster by at least an order of magnitude and as effective, if not more, in finding an optimal solution. Further, the speed and accuracy of the framework enables designers to perform sensitivity analyses on hyperparameters to determine the resiliency of the system to the changes of the hyperparameters.

1.1. Background and Related Work

In the era of the exigent need to design energy efficient neuromorphic systems for resource-constrained environments such as mobile edge devices, several approaches have been proposed in the literature to reduce the massive energy requirement of these systems. For artificial neural networks (ANNs), these techniques span from simplifying models, such as pruning and quantization (Han et al., 2015; Wen et al., 2016; Yang et al., 2018; Zoph et al., 2018), to designing energy efficient architectures (Jin et al., 2014; Panda et al., 2016; Parsa et al., 2017; Wang et al., 2017), and neural architecture search (Zoph et al., 2018). In spiking neuromorphic domain, these include different training algorithms such as Schuman et al. (2016), Bohnstingl et al. (2019) based on evolutionary optimization, Esser et al. (2015, 2016) on modified backpropagation techniques, Severa et al. (2019) as binary communication, and Rathi et al. (2020) as a hybrid approach and then deploying these on neuromorphic hardware such as Schmitt et al. (2017) and Koo et al. (2020). In this section, we briefly introduce each of these methods and continue with the added complexity of co-designing hardware and software for artificial neural networks and spiking neuromorphic systems. We then present the contribution of our work (Hierarchical-PABO) and how we fill the existing gap in a generic approach of co-designing hardware and software in the literature.

To reduce the energy requirement of neural network architectures, model simplification techniques proposed by Han et al. (2015), and continued with Wen et al. (2016), Zoph et al. (2018), and Yang et al. (2018). Each of these techniques focus on simplifying the neural network with different approaches of pruning, quantization, learning the connections, and leveraging sparsity. Designing energy-efficient architectures are also well-studied in the literature with flattened Convolutional Neural Network (CNN) (Jin et al., 2014), factorized CNN (Wang et al., 2017), conditional CNN (Panda et al., 2016, 2017), and staged-conditional CNN (Parsa et al., 2017). More recently, compact structures such as MobileNets (Howard et al., 2017) and ShuffleNet (Zhang et al., 2018) are also introduced and are specifically designed for mobile devices. Although both approaches of model simplification and efficient architecture design demonstrate promising results in reducing the energy requirements of neural networks, they do not necessarily yield to the optimum designs for energy efficient accelerators. This is mainly due to the fact that they only locally search the space. In addition, layers with more parameters do not necessarily consume more energy (Yang et al., 2017; Dai et al., 2019). Various techniques proposed for training spiking neural networks with different underlying hardware, are vital steps toward efficient neuromorphic computing for edge devices; however, each of these approaches require several hyperparameters and their optimum performance depend on prior knowledge on how to

set these hyperparameters. In Parsa et al. (2020), we showed that an optimum set of hyperparameters drastically increases the neuromorphic system performance.

There is a very rich literature on hyperparameter optimization and neural architecture search (NAS) techniques. Search for the optimum set of hyperparameters studied by Genetic CNN (Xie and Yuille, 2017), metaQNN (Baker et al., 2017), and SMBO (Liu C. et al., 2018). These techniques are built upon using Genetic algorithms or Bayesian optimizations. NAS was started by Google Brain (Zoph et al., 2018) to find an optimal neural architecture by searching for architectural building blocks on a small dataset and then transferring the block to larger ones. NAS was a starting point for a series of NAS-based approaches in recent years (Liu C. et al., 2018; Liu H. et al., 2018; Pham et al., 2018). All of these works were proposed to design a neural network with optimum performance, regardless of the energy requirement of the underlying neural accelerator.

Hardware-aware neural architecture designs can be categorized in three domains of multi-layer co-optimization (Reagen et al., 2016), hardware-aware NAS (Cai et al., 2018; Tan et al., 2019; Wu et al., 2019), and Bayesian-based hyperparameter optimization (Reagen et al., 2017; Marculescu et al., 2018; Stamoulis et al., 2018). Each one of these approaches have their pros and cons. While defining an optimum neural architecture with energy-efficient hardware in mind, the multi-layer co-optimization approach cannot easily be extended to generic platforms. Hardware-aware NAS techniques are time consuming and require substantial resources, and Bayesian-based methods are not well-suited for parameter-heavy models Dai et al. (2019). In Hierarchical-PABO, we propose a novel hardware-aware approach with minimum mathematical complexity. This framework is based on hierarchical Bayesian optimization and agent-based modeling. Using a set of Bayesian estimators in different levels and correlating them using a supervisor agent, we overcome the drawbacks of exclusive Bayesian approaches available in the literature.

1.2. Main Contributions

We made the following contributions:

1. **A novel optimization framework based on hierarchical Bayesian optimization and agent-based modeling, suitable for both artificial neural networks and spiking neuromorphic systems.** With simple yet effective underlying mathematics, Hierarchical-PABO estimates the Pareto region for multi-objective hyperparameter optimization problems with few evaluations.
2. **One of the first techniques in the literature for co-designing software-hardware that is not limited to the number of objectives to optimize (network performance, energy consumption, size, speed of inference, etc.).** Based on our knowledge, our proposed technique is one of the first techniques in the literature that simplifies the mathematical complexity of exclusive Bayesian approaches for multi-objective optimization. We do this by adding a supervisor agent and performing Bayesian optimization in different

levels. This paves the way to effectively optimize more than two objective functions.

3. **Generic framework extendable to various artificial and spiking neural networks and the underlying digital, analog, or mixed-signal accelerators.** We tested our framework on several classification and control applications on digital and mixed-signal accelerators and were able to estimate the Pareto frontier regardless of the size of the search space.
4. **Superior performance in terms of accuracy and computational speed compared to the state-of-the-art Genetic Algorithm (GA) optimization approach** (in scenarios where GA-based optimizations were available for comparison, Deb et al., 2002). Please see Parsa et al. (2019a) for details of this contribution.

2. METHODOLOGY AND EXPERIMENTAL SETUP

In order to systematically take the human knowledge out of the loop in selecting the optimum set of hyperparameters for a neuromorphic system (and in general any artificial intelligence-based computing system), we chose Bayesian optimization as the core of our approach. In this section, we first overview the basic mathematics of Bayesian modeling and justify the use of this technique in our proposed Hierarchical Pseudo Agent-based Bayesian Optimization (Hierarchical-PABO) framework, and then present the experimental setup for this approach.

2.1. An Introduction to Bayesian Optimization

Bayesian optimization is a powerful tool for finding the optimum point of objective functions that are unknown and expensive to evaluate (Brochu et al., 2010). The problem of finding a global optimizer for an unknown objective function is formulated in Equation (1).

$$x^* = \operatorname{argmax}_{x \in X} f(x) \quad (1)$$

where X is the entire design space, and f is the black-box objective function without simple closed form. As summarized by Shahriari et al. (2015), in a sequential manner, we search for the best location x_{n+1} to observe y_{n+1} point in order to estimate f . After N iterations, the algorithm suggests the best estimation of the black-box function f . This sequential approach is based on building a prior estimation over possible objective functions, and then iteratively re-estimating the prior model using the observations from updating the Bayesian posterior model. The posterior representations are the updated knowledge on the objective function we are trying to optimize. We explore the search space by leveraging the inherent uncertainty of the posterior model and mathematically introducing a surrogate model, called the acquisition function α_n . The maximum point of this function is the next candidate point to observe (x_{n+1}) and guides the search direction toward the true representation of the objective function. The efficiency of Bayesian approach to estimate the global optimizer for the expensive black-box function with fewer evaluations lies on the ability of Bayesian

technique to learn from prior belief on the problem and direct the observations by trading off exploration and exploitation of the design space.

In the context of neuromorphic computing, x is the system's hyperparameters such as inherent hyperparameters for different input/output encoding schemes, or population size or optimizer choice for various training techniques. Hardware-specific hyperparameters are also another choice for parameter x . Function f is the black-box objective function, such as accuracy of the network, energy or area requirements of the system, and speed of inference, for stochastic observations of y . A summary of the Bayesian approach is illustrated in the **Figure 1**. See Brochu et al., 2010; Bergstra et al., 2011; Eggenberger et al., 2013 for detailed tutorials.

In **Figure 1**, we are estimating an unknown objective function, ground truth f . We only have two observations (likelihood model) in iteration one (red dots). We first build our prior distribution (current belief) based on these observations using Gaussian processes. The Gaussian distribution is shown with mean and standard deviation, solid black line, and highlighted dashed area, respectively. A surrogate model, acquisition function, is estimated for this posterior distribution, which is shown as the highlighted green function. The maximum point of the acquisition function (green dot) is the best next point to observe in the next iteration. As the new points are added to the observations in different iterations, the standard deviations, and therefore the uncertainty of estimating the ground truth function, is reduced. Each observation requires evaluating an unknown, expensive objective function. The ability of the Bayesian technique in predicting this function (ground truth in **Figure 1**) with few evaluations, speeds up the process of finding the optimum set of hyperparameters with minimum computational resources.

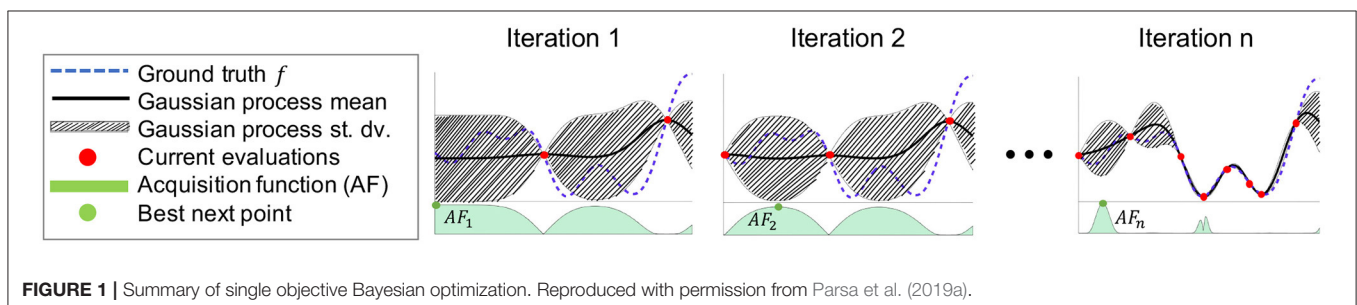
For configuring the Gaussian process, the covariance function is a positive definite kernel that specifies the similarity between points of observations. There are different methods to estimate this kernel function based on the smoothness, noise level and periodicity of the ground truth. In our experimental setup, we selected the Matern kernel function with smoothness value of 1.5. This particular kernel is selected due to the intrinsic stochastic nature, and noise level of our problem. Once we estimate the posterior distribution based on the likelihood model and the prior distribution, we build an acquisition function to guide the search direction. This acquisition function defines whether to search the space where the uncertainty is high

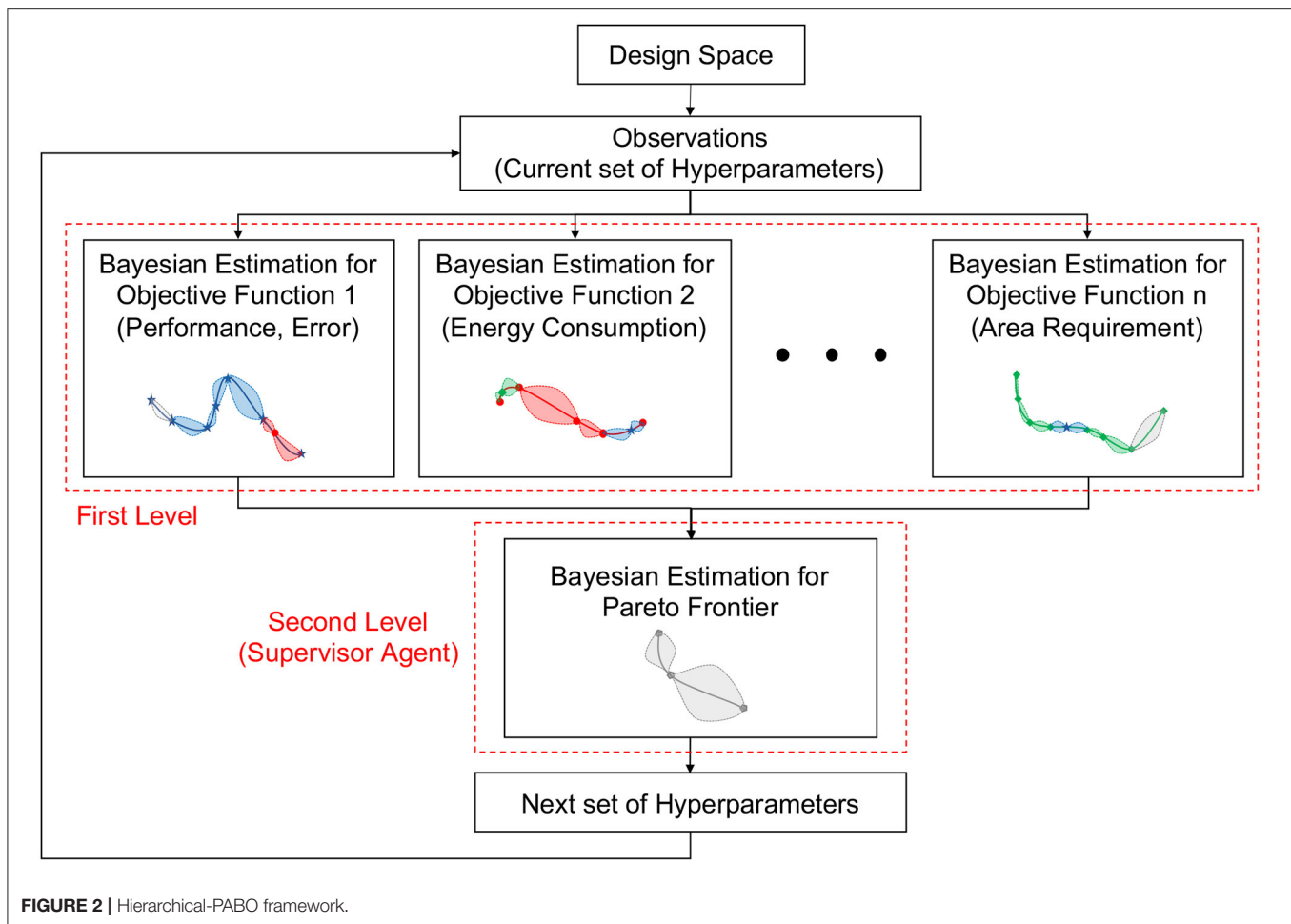
(explore) or sample at locations where the model predicts high objectives (exploit). There are different methods to calculate this surrogate model (Kushner, 1964; Lai and Robbins, 1985; Jones et al., 1998; Jones, 2001; Brochu et al., 2010; Bull, 2011; Agrawal and Goyal, 2013; Hernández-Lobato et al., 2014). The choice of the method to use directly impacts the speed of convergence to the ground truth in Bayesian search. We chose expected improvement approach for the acquisition function. This selection does not impact the effectiveness or performance of our approach; rather, it only impacts the speed of searching the hyperparameter space and avoid trapping in local minima. More details in selecting kernel or acquisition function can be found in Shahriari et al. (2015).

2.2. Hierarchical-PABO

Hierarchical-PABO (Hierarchical Pseudo Agent-based Bayesian Optimization) is an ultra-efficient Bayesian-based optimization framework to find an optimum set of hyperparameters for designing an accurate neural network while minimizing energy consumption and area requirement of the underlying hardware.

Figure 2 summarizes the Hierarchical-PABO framework. We randomly select two hyperparameter (HP) combinations from the design space. In the first level, these current observations are used to build Bayesian estimation posterior distributions for each objective function separately. We then define the acquisition function for each posterior model. The optimum point of these acquisition functions are the best next point (HP combination) to evaluate for their corresponding objective function. In the second level, the supervisor agent level, the process starts with all current observations (set of HP combinations) and the candidate HP combination that led to the optimum value of the acquisition functions in the previous iteration. For these observations, we estimate an intermediate Pareto frontier function using a Gaussian distribution. This is calculated based on the observation points (on the Pareto front set), as well as a score calculated based on L1-norm of these points after being normalized. Therefore, a corresponding surrogate model (acquisition function) for this Gaussian distribution explores and exploits the search space with the goal of estimating the current intermediate Pareto function. The next best observation for this Pareto is then added to the observations for each Bayesian estimator. With this technique, we force the Bayesian approach to add extra observations that help in minimizing the current intermediate Pareto function. This function is





updated iteratively and moved toward the actual Pareto region of the problem.

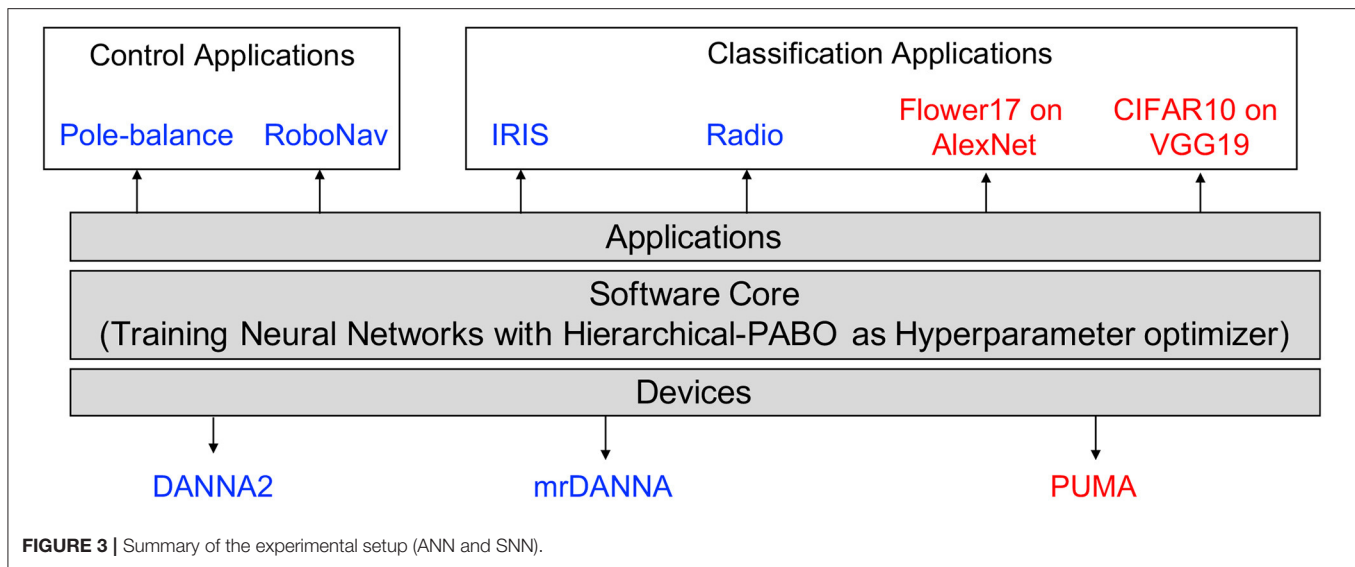
In Hierarchical-PABO, the Pareto Bayesian estimator in the second level plays a vital role in correlating the Bayesian estimators for each objective function in the first level. However, to speed up the search process, the supervisor agent might turn off this Pareto Bayesian estimator. If this extra Bayesian estimator is turned off, the supervisor agent takes HP combinations taken from optimum point of the acquisition function for each objective and only allow those that are in favor of moving toward the Pareto region. Please see the **Supplementary Material** for Hierarchical-PABO pseudo-code.

2.3. Experimental Setup

An overview of our experimental setup is shown in **Figure 3**. We test Hierarchical-PABO on several devices for various control and classification tasks. For experiments on Artificial Neural Networks (ANNs), we select PUMA (Ankit et al., 2019) as the underlying hardware with two different deep neural network architectures, AlexNet (Krizhevsky et al., 2012) and VGG19 (Simonyan and Zisserman, 2014) on Flower17 (Nilsback and Zisserman, 2006), and CIFAR10 (Krizhevsky, 2009) image classification dataset.

For Spiking Neural Networks (SNNs), we consider both digital and mixed-signal hardware; DANNA2 (Mitchell et al., 2018), and mrDANNA (Chakma et al., 2017), respectively. Additionally, we select Pole-balance (Wieland, 1991; Gomez et al., 2006), and RoboNav (Mitchell et al., 2017) for experiments on control applications, and IRIS (Dua and Graff, 2017), and Radio (Reynolds et al., 2018) dataset for classification applications. In **Figure 3**, the experimental setup for ANN is shown in red, and for SNN in blue.

In SNN domain, we utilize a modified version of the TENNLab neuromorphic software framework (Plank et al., 2017, 2018). This platform enables studying different applications and evaluating them on several neuromorphic processor implementations. This capability is well-suited for the purpose of our hyperparameter multi-objective optimization as it allows switching applications and devices within the framework without the need to change the software. We modify the TENNLab framework by adding Hierarchical-PABO to its primary underlying learning algorithm, which is Evolutionary Optimization for Neuromorphic Systems, EONS (Schuman et al., 2016). EONS is an evolutionary approach for designing the network topology and parameters of an SNN for a given application and neuromorphic hardware implementation. This



evolutionary algorithm follows the same steps as a traditional evolutionary approach. That is, EONS begins with a population of potential solutions and evaluates each of those solutions on the problem at hand (running the potential network solution on the application on the hardware or a simulation of the hardware) to get a fitness score for each solution. Then, EONS uses the fitness scores to perform selection (preferentially selecting better performing networks to serve as parents) and reproduction (to produce children networks from the parents). Reproduction includes both crossover operations (taking components from two networks to assemble one or more children), mutation operations (small-scale changes such as parameter updates or adding or deleting a neuron or synapse), and duplication. Details of the experimental setup for both ANN and SNN are described in this section.

2.3.1. Experimental Setup for ANN

For experiments in ANN domain, to speed up the search for the optimum hyperparameter, we turn off the extra Bayesian estimator block in the supervisor agent. In this case, the supervisor agent only correlates the results of the isolated Bayesian estimations of each objective function, and decides on the best hyperparameter combination for the next iteration based on the ones that might belong to the Pareto frontier. Details of the Hierarchical-PABO when the extra Bayesian estimator in supervisor agent is turned off is given in Parsa et al. (2019a).

As discussed in Parsa et al. (2019a), the underlying hardware we select for our ANN experimental setup is a programmable ultra-efficient memristor-based accelerator called PUMA, proposed by Ankit et al. (2019). This spatial general-purpose architecture is based on hybrid CMOS-memristor technology that enables mapping machine learning applications using on-chip memory only. Analog memristor crossbars, functional units, and instruction execution pipelines are the building blocks of PUMA's core. Multiple cores create tiles via a shared memory. PUMA's nodes are several tiles connected

through an on-chip network. For large-scale executions, PUMA nodes are linked with a chip-to-chip interconnect.

To calculate energy consumption of PUMA, we use an abstract energy consumption model of the memristor crossbars only. This enables evaluating the impact of hyperparameters on the energy usage of PUMA, while isolating the benefits of micro-architectural design. We expect lower energy usage with less number of crossbars. Details of calculating the energy consumption of PUMA is given in Equation (2).

$$\begin{aligned} \text{Total Energy} = & \left[\sum_i (d_i \times d_i \times \lceil \frac{nc_i \times k_i \times k_i}{xs} \rceil \times \lceil \frac{nc_{i+1}}{xs} \rceil) \right. \\ & \left. + \sum_i (\lceil \frac{nf_i}{xs} \rceil \times \lceil \frac{nf_{i+1}}{xs} \rceil) \right] \times ep\text{x} \end{aligned} \quad (2)$$

In Equation (2), the total energy consumption is the summation of number of crossbars needed for all convolution and fully connected layers multiplied by the energy per matrix vector multiplication operation ($ep\text{x}$). In PUMA's memristive crossbar accelerator, $ep\text{x}$ is $\simeq 44$ nJ for a 16-bit (inputs and weights) crossbar operation with crossbar size (xs) of 128×128 . For the i_{th} convolution layer, d_i is the dimension of the output, nc_i is the number of input features, and k_i is the kernel size. The dimension of the output in the convolution layer is for the inherent weight-sharing property of these layers. For the i_{th} fully connected layer, nf_i is the number of input features.

In the ANN's experimental setup, we used AlexNet (Krizhevsky et al., 2012) and VGG19 (Simonyan and Zisserman, 2014) for the deep neural network architectures. For details on the structures of AlexNet and VGG19 please refer to the **Supplementary Material**. We performed several case studies for different types of hyperparameters, including the number of layers, kernel sizes, number of features to extract in each layer, and also the values for learning rate, momentum, and dropout. The details of the our proposed

TABLE 1 | Energy estimate per spike for mrDANNA.

	Accumulation	Fire	Learning	Idle
Neuron	9.81pJ	12.5pJ	-	7.2pJ
Synapse	1.45pJ	-	2.58pJ	0.07pJ

hyperparameter optimization technique on ANN results are given in Parsa et al. (2019a).

2.3.2. Experimental Setup for SNN

As mentioned in section 2.2, based on the complexity of the problem, the supervisor agent decides to keep the extra Bayesian estimator block on or off. In SNN domain, this block is turned on which is well-suited for the hyperparameter optimization of spiking neuromorphic systems. In these systems, the intrinsic HPs in different building blocks of these systems are so critical in the final performance of the system that an additional Bayesian optimizer is needed to find the optimum set of HPs.

The summary of the applications and neuromorphic processors we select for SNN experimental setup is shown in **Figure 3**. For the applications, we tested Hierarchical-PABO on both control and classification tasks. Pole-balance (Wieland, 1991; Gomez et al., 2006), and RoboNav (Mitchell et al., 2017) were the two selected control applications. Pole-balance is a control benchmark in engineering which involves a pole connected to a cart through a joint that allows single axis movement. The goal of this control application is to keep the pole from falling by moving the cart either direction. RoboNav is an autonomous navigation system for robotic applications and is meant to be deployed on a specific robot (Mitchell et al., 2017). We also used the Iris (Dua and Graff, 2017) and Radio (Reynolds et al., 2018) datasets for classification tasks. The former is a multivariate dataset of 50 samples from each of three species of the Iris flower, and the latter is a satellite radio signal classification problem.

We use two different neuromorphic implementations that are already deployed in the TENNLab framework, a fully digital neuromorphic processor, DANNA2 (Mitchell et al., 2018), and a memristive mixed-signal neuromorphic processor, mrDANNA, (Chakma et al., 2017). DANNA2 is a fully digital programmable device with integrate-and-fire neurons and synapses, and mrDANNA is a mixed analog-digital programmable device with metal-oxide memristors. We use mrDANNA for the case studies where we would like to minimize energy requirement of the underlying neuromorphic hardware. **Table 1** summarizes the energy estimate per spike for this neuromorphic device. mrDANNA is a synchronous neuromorphic architecture and is simulated in a discrete event simulation. Events in the simulation include accumulations, fires, and learning. The energy estimates for each event type are given in **Table 1** and we track how many of each type of event occurs in the simulation and sum up the energies. If no event is occurring on a neuron or synapse in a clock cycle, that neuron or synapse is “idle,” but still performing some operations that contribute to idle

cost. We use these energy estimates to estimate the overall energy cost of running on a particular application.

3. RESULTS

To validate Hierarchical-PABO we consider different case studies, which are summarized in **Table 2**. Different applications (control and classification), architectures (AlexNet and VGG19 for ANN, and EONS for SNN), dataset (Flower17, CIFAR10, IRIS, Radio, and Pole-balance), and accelerators (PUMA, DANNA2, mrDANNA) are considered with different search space sizes. These different case studies are chosen to demonstrate our proposed generic hyperparameter optimization approach.

3.1. Results for ANN

Table 3 shows a summary of the selected ranges for the hyperparameters (HPs) for each ANN case study given in **Table 2**. All these cases are studied with PUMA as the underlying hardware. Case study one is designed with a small search space of size 192 HPs. We begin with the small search space size in order to estimate the actual Pareto frontier of the problem with a grid search technique and to compare the Hierarchical-PABO (H-PABO) result with other state-of-the-art approaches. Case study two is included to capture the effects of different types of HPs in the analysis, and case study three is a more realistic experiment with VGG19 as the chosen architecture on CIFAR10 dataset.

Figure 4 demonstrates results for different case studies. Each point in this figure corresponds to a set of HPs from the ranges given in **Table 3**. H-PABO search points are shown in red circles and are the selected HP combinations that lead to defining a Pareto frontier region. As already discussed in section 2.2, this selection is based on exploring and exploiting the search space. In all three case studies shown in **Figure 4**, the H-PABO search not only emphasizes on the Pareto region, but also explores the search space to avoid trapping in local minima.

In **Figure 4A**, H-PABO search points are compared to grid search (shown in gray crosses), random search (blue diamonds), and state-of-the-art NSGA-II (Deb et al., 2002) search (black squares). H-PABO predicts the actual Pareto frontier of the problem with only 17 evaluations (out of 192 possible HP combinations). This result outperforms other approaches not only in accuracy of predicting the Pareto frontier, but also in superior computational speed. The random search results are from 40 evaluations of HP combinations, and NSGA-II is based on a population size of 10 with a maximum generation of 50. In this case study, H-PABO is 92× faster than NSGA-II in predicting the actual Pareto frontier of the problem. An optimal design that belong to the Pareto frontier with 26% error and 7mJ PUMA energy consumption will lead to almost 40% decrease in energy consumption compared to a not-optimal design with 26% error and 12mJ energy consumption. For these two designs all hyperparameters such as dropout, learning rate, and optimizer type are similar, except number of fully connected layers, convolution layers, and two filter sizes. The optimal design has two fully connected layers, and four convolution layers with filter sizes 3 in the second and third layers. However, the

TABLE 2 | Case studies for hierarchical-PABO.

Case study	Domain	Application	Architecture	Dataset	Accelerator	Search space	Objective
One	ANN	Classification	AlexNet	Flower17	PUMA	192	Accuracy, Energy
Two	ANN	Classification	AlexNet	Flower17	PUMA	288	Accuracy, Energy
Three	ANN	Classification	VGG19	CIFAR10	PUMA	3,072	Accuracy, Energy
Four	SNN	Control	EONS	Pole-Balance	DANNA2	240	Accuracy
Five	SNN	Control	EONS	Pole-Balance	DANNA2	54,432,000	Accuracy
Six	SNN	Classification	EONS	IRIS	mrDANNA	1,458	Accuracy, Energy, Size
Seven	SNN	Classification	EONS	Radio	mrDANNA	1,458	Accuracy, Energy, Size
Eight	SNN	Classification	EONS	IRIS	mrDANNA	35,460	Accuracy, Energy, Size
Nine	SNN	Classification	EONS	Radio	mrDANNA	35,460	Accuracy, Energy, Size

TABLE 3 | Evaluated parameters for three different case studies for ANNs.

	Case study one	Case study two		Case study three
Dropout	0.4, 0.5	0.5	Dropout, Layer 1	0.3, 0.4
Learning Rate	0.001	0.001, 0.01	Learning Rate	0.01, 0.1
Momentum	0.85, 0.9, 0.95	-	Learning Rate Decay	1e – 6, 1e – 4
Optimizer	Momentum	Momentum, Adam	Weight Decay	0.0005, 0.05
# of FC Layers	2, 3	2, 3	Kernel Size, Layer 6	3, 5
# of Conv. Layers	4, 5	3, 4, 5	Kernel Size, Layer 7	3, 5
Kernel Size, Layer 1	5, 7	3, 5, 7	Kernel Size, Layer 8	3, 5
Kernel Size, Layer 2	3, 5	3, 5	Kernel Size, Layer 9	3, 5, 7
Kernel Size, Layer 3	3, 5		# of Features, Layer 1	64, 128
Kernel Size, Layer 4	3	3, 5	# of Features, Layer 2	128, 256
			# of Features, Layer 4	256, 512
Architecture	AlexNet	AlexNet		VGG19
Neural Accelerator	PUMA	PUMA		PUMA
Dataset	Flower17	Flower17		CIFAR10
Search Space	192	288		3072

not-optimal design has three fully connected layers, and five convolution layers with filter sizes 5 in the second and third layers. Further analysis on the results is given in Parsa et al. (2019a).

For case study two given in **Table 3**, we show the convenience of changing HP types within the H-PABO framework by incorporating the choice of optimizer as an HP. In **Figure 4B,C**, H-PABO estimates the Pareto region with 39 and 22 evaluations, respectively. The complexity and predictability of the problem upon changes of HP combinations define the speed of H-PABO in predicting the Pareto region.

3.2. Results for SNN

Table 4 shows a summary of the selected ranges for the hyperparameters (HPs) for case studies in SNN domain given in **Table 2**. In this table, b_k , p_k , $[c_k, C_k]$, *function*, and *interval* are from the input encoding module, *population size*, *mutation rate*, and *crossover rate* are for EONS evolutionary-based training algorithm, and *synaptic weight*, *neuron threshold*, and *synaptic delay* belong to the underlying neuromorphic hardware. The input encoding hyperparameters include several approaches such as *binning-based*, using b_k as the number of bins required for each input values, *spike-count* with p_k as the maximum number

of spikes to encode a single input value, *charge-value* with $[c_k, C_k]$ on injecting a specific charge to fire a neuron, *function* on how to map the values to spikes, and *interval* to define the interval between pulses. For more details on each of these hyperparameters please refer to Parsa et al. (2019b), Schuman et al. (2019).

We first show the importance of hyperparameter optimization for spiking neuromorphic systems by only focusing on single-objective optimization (performance of the system on the task) problem, where grid search results are already available by Schuman et al. (2019). We then continue with Hierarchical-PABO (H-PABO) results for a three-objective optimization problem (performance, energy, and network size).

Single-Objective Optimization with Hierarchical-PABO (H-PABO): While H-PABO is generally aimed for multi-objective problems, it can easily be reduced to a single-objective optimization by setting objective functions to one. This is the case for case study four, where we are only optimizing a single objective function that is the accuracy of the neural network. The details of this case study is given in **Table 4**. **Figure 5** shows box plot figures with interquartile ranges. The grid search result is produced and published by Schuman et al. (2019) and shown in **Figure 5A**. For each one of the 240 combinations

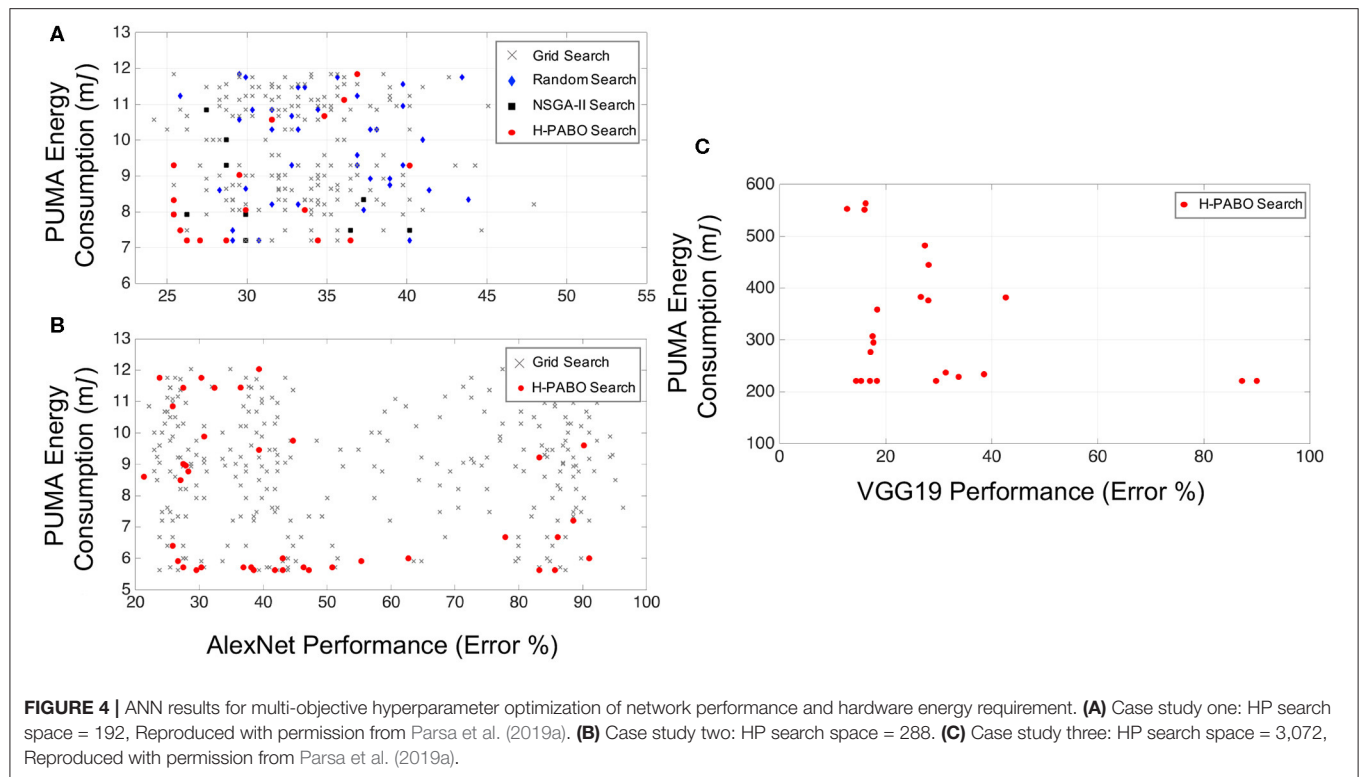


TABLE 4 | Evaluated parameters for case studies four to nine for SNNs.

Hyperparameters	Case study four	Case study five	Case studies six, and seven	Case studies eight, and nine
b_k	1, 2, 4, 8	2, ..., 8	2, 4, 8	2, 4, 8, 10, 12
ρ_k	1, 2, 4, 8	1, ..., 12	4, 8	2, 4, 8, 10, 12
$[C_k, C_k]$	[0,0.5],[0,1], [0.25,0.5], [0.25,1], [0.5,0.5],[1,1]	[0,0.5],[0,1], [0.25,0.5],[0.25,1], [0.5,0.5],[1,1]	[0,1], [0.5,0.5], [1,1]	[0,0.5],[0,1], [0.25,0.5], [0.25,1], [0.5,0.5],[1,1]
Function	simple, flip-flop, triangale	simple,flip-flop,triangale	simple, flip-flop	simple, flip-flop, triangale
Interval	1	1, ..., 5	0, 1	0, 1, 2
Population size	1,000	600, 800, 1,000, 1,200, 1,500, 2000	10, 100, 500	10, 100, 500, 700
Mutation rate	0.9	0.6, 0.7, 0.8, 0.9	0.2, 0.6, 0.9	0.2, 0.6, 0.9
Crossover rate	0.5	0.3, 0.4, 0.5, 0.6, 0.7	0.3, 0.5, 0.9	0.3, 0.5, 0.9
Synaptic weight	[-255,255]	[-127,127],[-255, 255] [-511, 511],[-1023, 1,023]	-	-
Neuron threshold	[0,1,023]	255, 511, 1023	-	-
Synaptic delay	127	15, 31, 63, 127, 255	-	-
Neural Accelerator	DANNA2	DANNA2	mrDANNA	mrDANNA
Application	Pole-balance	Pole-balance	six: IRIS seven: Radio	eight: IRIS nine: Radio
Search Space	240	54,432,000	1458	35,640

of the hyperparameters, the network accuracy is calculated and evaluated for 100 times. In **Figure 5B**, we used H-PABO for the same experiment, and with only 40 hyperparameter

combinations, each repeated for 10 times, we are able to predict not only the exact optimum set of hyperparameter, but also predict the same trend in the network accuracy changes for

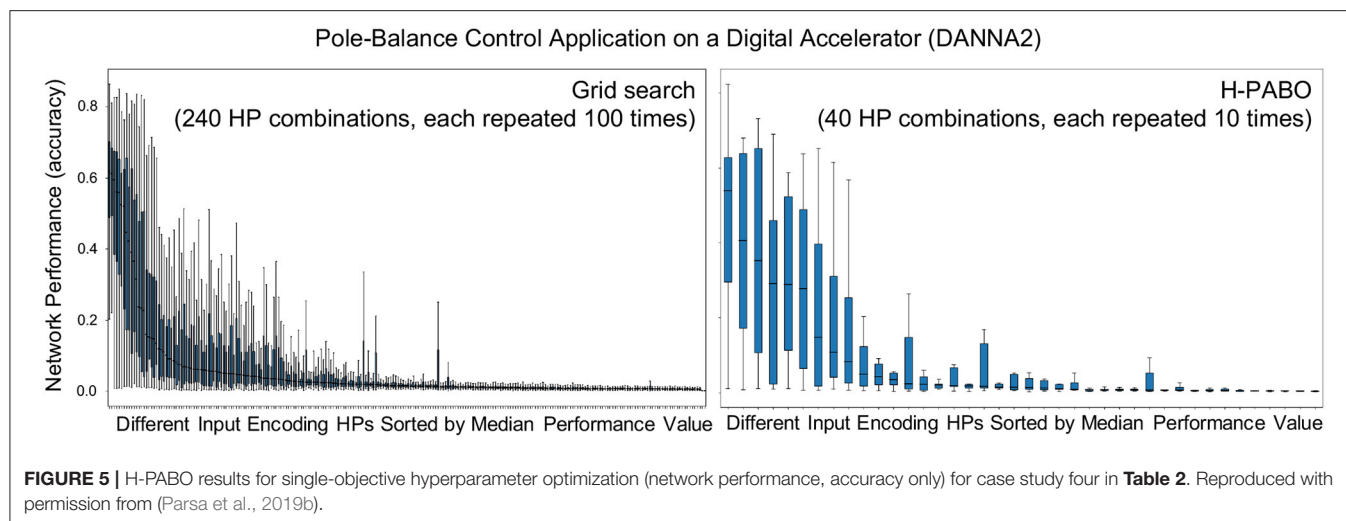


TABLE 5 | Sensitivity analysis for H-PABO single objective optimization.

	HPs	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Input encoding HPs	b_k	2	2	2	2
	ρ_k	8	12	8	8
	Charge	[0, 0.5]	[0, 0.5]	[0, 0.5]	[0, 0.5]
	Function	Flip-flop	Flip-flop	Flip-flop	Flip-flop
	Interval	1	5	1	2
EONS HPs	Population size	1000	1500	400	1000
	Mutation rate	0.9	0.9	0.9	0.9
	Crossover rate	0.5	0.4	0.5	0.7
	Synp weight	[-255, 255]	[-127, 127]	[-255, 255]	-
Accelerator HPs	Neuron threshold	[0, 1023]	[0, 1023]	[0, 1023]	-
	Synp delay	127	255	15	-
Neuromorphic System Performance		52%	70.99%	50%	53%
Accelerator		DANNA2	DANNA2	DANNA2	mrDANNA
Application		Pole-Balance	Pole-Balance	RoboNav	RoboNav

different hyperparameter combinations (Parsa et al., 2019b). In this case study the optimum hyperparameter combination leads to median value of 52%.

The hyperparameters are kept exactly similar between case studies four and five in **Table 4**. However the ranges for each hyperparameter is increased in case study five. Although all hyperparameters are still in reasonable ranges, the search space is drastically increased to over 54 million different hyperparameter combination in case study five. This shows that in real problems where different hyperparameters exist originating from different modules of the system such as input encoding, hardware, or the training algorithm itself, hyperparameter optimization plays vital role in obtaining the maximum performance of the system. We performed H-PABO to define the set of hyperparameter that optimizes network's accuracy and were able to increase the median value of the accuracy to 70.99% compared to 52% in case study four. Please refer to Parsa et al. (2019b) for more details on single-objective hyperparameter optimization on spiking neural networks.

In **Table 5**, a sensitivity analysis is performed for H-PABO single objective optimization for different classification applications on two different neural accelerators. These experiments show how sensitive is pole-balance control application to the changes of hyperparameters. If we only change few hyperparameters (all in reasonable ranges), the resulting accuracy will change from 52 to 70.99% (comparing experiments 1 and 2 in **Table 5**). Based on these experiments, RoboNav appears to be less sensitive to changes in hyperparameters and architectures, but more extensive experiments may be required in order to understand the full impact on this particular application.

Three-Objective Optimization with Hierarchical-PABO (H-PABO): To validate H-PABO technique for multi-objective hyperparameter optimization problems in SNN domain, we focus on classification application with IRIS (Dua and Graff, 2017), and Radio (Eggenberger et al., 2013) dataset on both digital (Mitchell et al., 2018), and mixed-signal memristive (Chakma et al., 2017)

neuromorphic devices. The summary of the case studies six to nine, and their corresponding HP ranges are given in **Tables 4, 2**, respectively.

Figure 6 demonstrates the Hierarchical-PABO (H-PABO) results in SNN domain on IRIS classification dataset on a mixed-signal underlying hardware [mrDANNA, Chakma et al. (2017)].

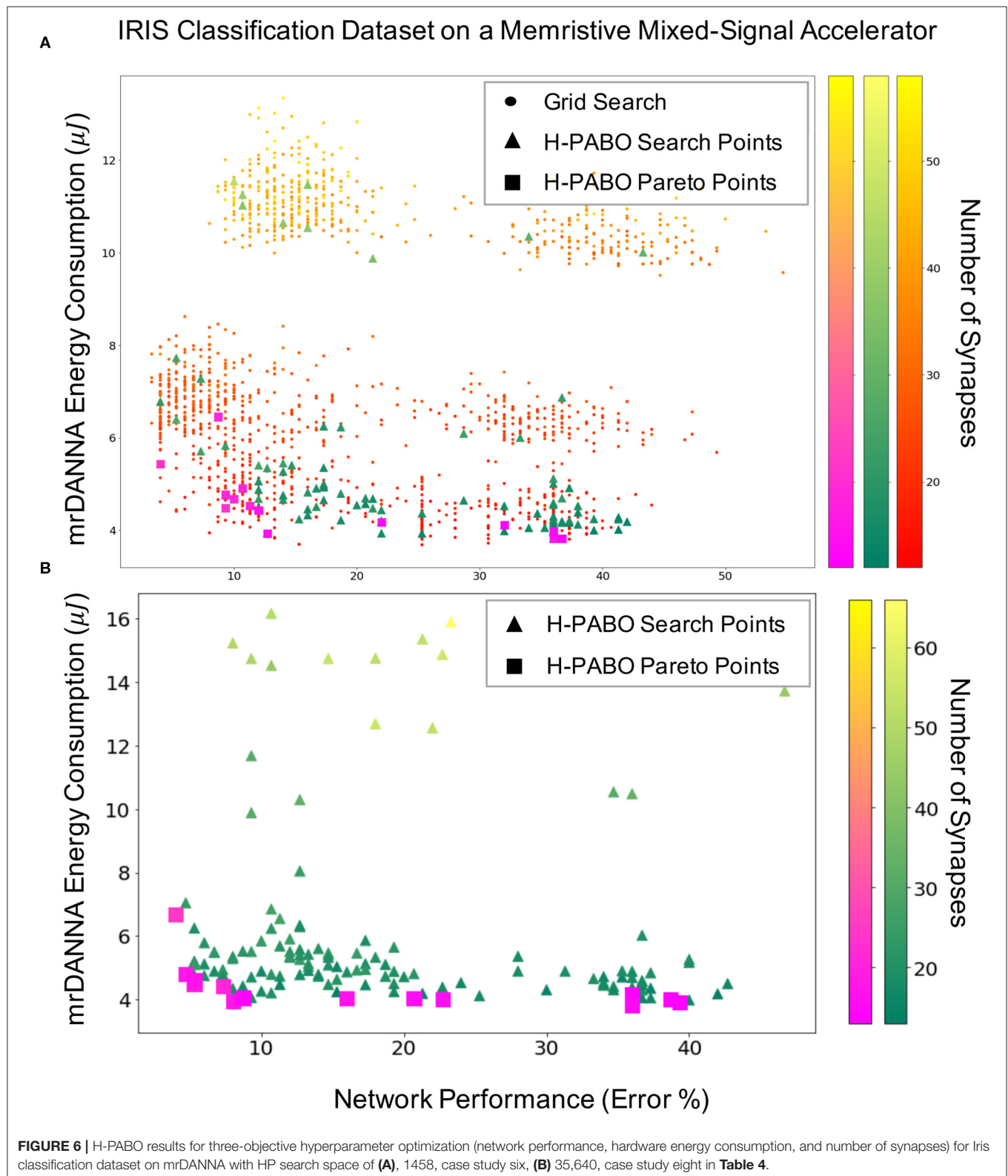


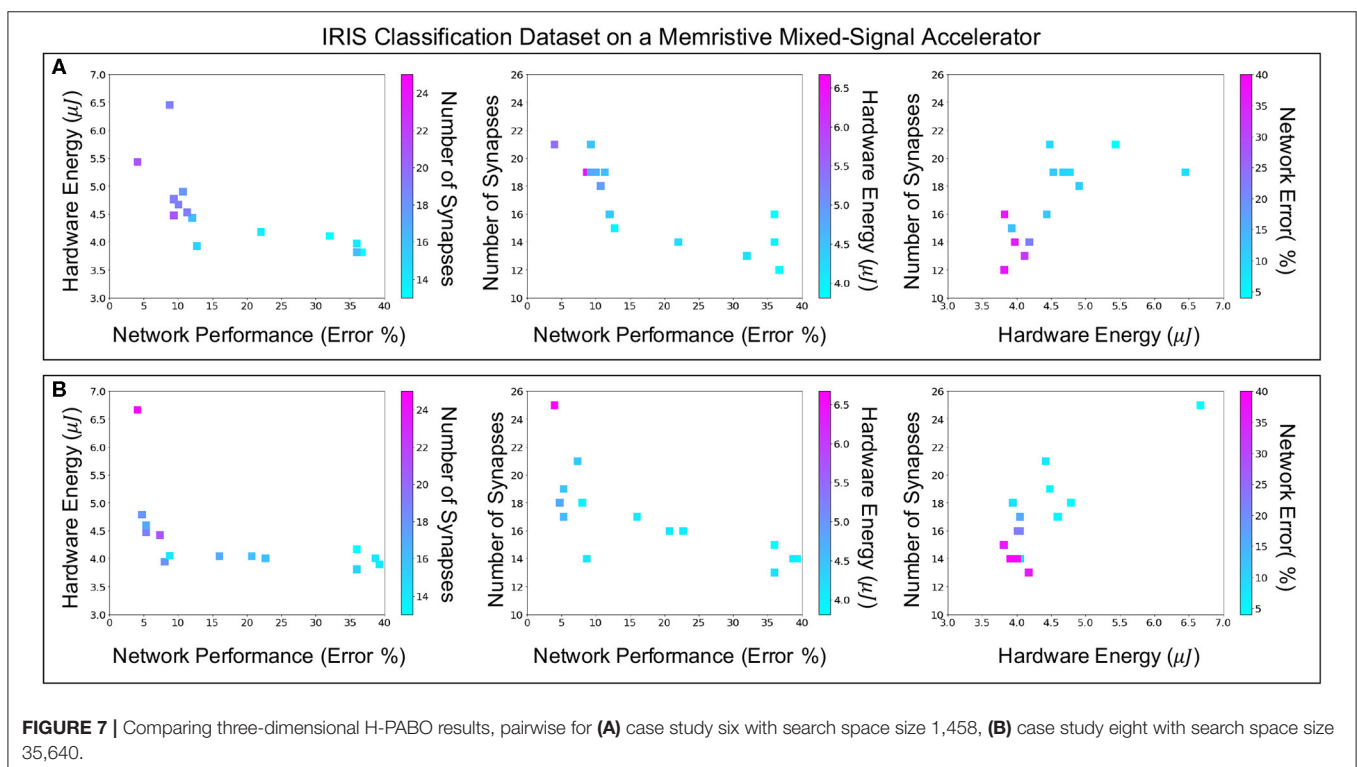
Figure 6A shows the H-PABO results compared to grid search for the case study six given in **Table 4** with 1458 different sets of HPs. Each point in the three-dimensional figure represents network performance, hardware energy consumption, and number of required synapses for a set of HP combination. The number of required synapses increases as the color becomes lighter. The grid search results show that most of the time the energy consumption increases as the number of synapses increase (the top left region of **Figure 6A**). However, we might also have a larger network with more inhibitory synapses, for example, that would have less activity and thus less energy than a smaller network (top right region). The triangles are the H-PABO search points, and as expected, all different regions of the search space are explored with H-PABO. The H-PABO Pareto points are shown with squares. These points are calculated once the H-PABO search process is completed and are the H-PABO search points that belong to the Pareto frontier. As shown in **Figure 6A** this calculated Pareto frontier is within close proximity to the actual Pareto frontier of the problem.

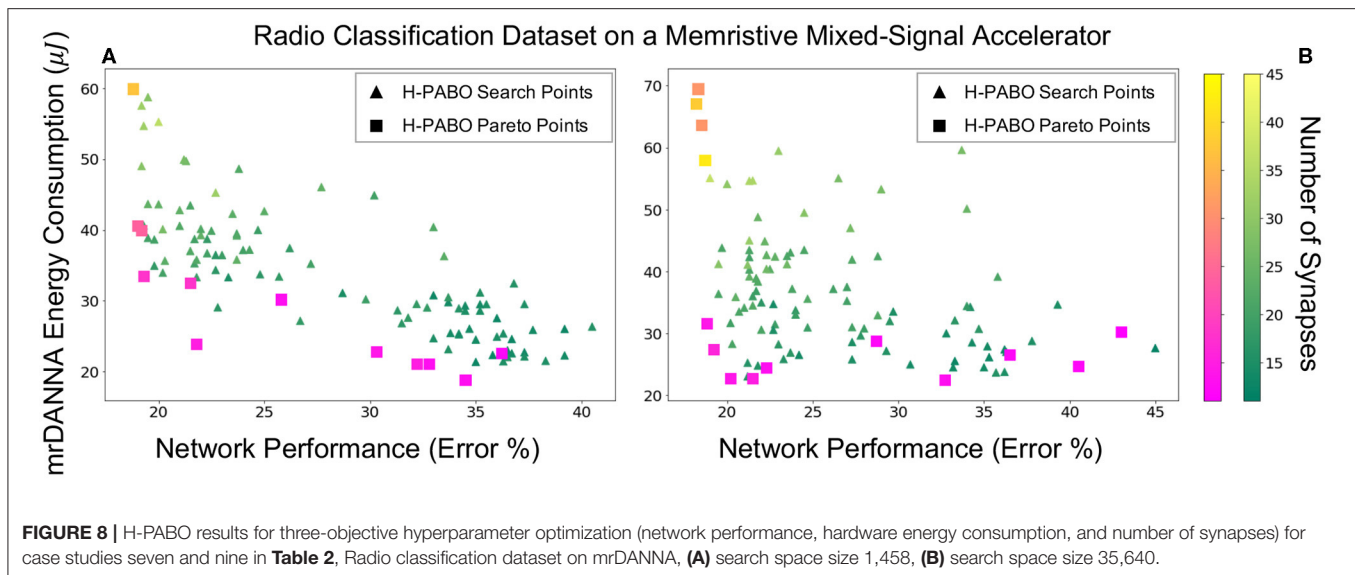
Figure 6B shows the H-PABO results for case study eight in **Table 4** for the HP search space of 35640 different HP combinations. Once again, we see that all regions of the search space are explored by the H-PABO approach, but that the majority of the H-PABO points are evaluated are in the region of interest and near the H-PABO Pareto front. In this case, H-PABO was able to find well-performing networks with desired characteristics (low energy consumption and relatively few synapses) with significantly fewer evaluations than what would be required for a full grid search of 35,640 points. It is also worth

noting that by optimizing over the additional HPs, the H-PABO approach is able to find well-performing networks with better characteristics than the networks found simply optimizing over the smaller set of HPs (shown in **Figure 6B**).

Figure 7 shows the H-PABO results from **Figure 6**, but splits the results into three different pairwise comparison plots, for each case study, to show how the different objectives play off of each other. The third objective is also shown in each plot through the color of the squares. With these plots, we can see the different Pareto fronts for each of the pairwise objectives. For example, in the network performance vs. hardware energy plots, we can see that there are trade-offs in energy usage in order to achieve lower error (and similarly for network performance vs. number of synapses). However, the number of synapses and energy usage are relatively correlated, such that fewer synapses typically corresponds to a lower energy value.

Figure 8 gives the results for case studies seven and nine, in which the H-PABO approach is applied to HP optimization for the Radio classification dataset on the memristive mixed-signal system (mrDANNA). The two case studies look at the same HP combination sets as the Iris dataset and correspond to 1458 and 35640 combinations, respectively. As we can see in the figure, H-PABO once again explores the space of potential solutions but is able to find a Pareto front in relatively few evaluations. Again, similar to the result for the Iris dataset, we can see that by expanding our HP set to the 35640 potential HP combinations, H-PABO is able to achieve overall better performing networks (lower error and energy and fewer synapses required), and in general moving the Pareto front closer to the desired region.





4. DISCUSSION AND FUTURE WORK

In this paper, we propose a novel multi-objective optimization framework based on hierarchical Bayesian optimization and agent-based modeling (Hierarchical-PABO). With its one of a kind structure, and simple yet effective underlying mathematics, we are able to predict a Pareto frontier of a multi-objective hyperparameter optimization for both non-spiking and spiking neural network systems with only few evaluations. This framework paves the way to further analyze and study sensitivity and resiliency of the system due to the changes of the hyperparameters.

The main current limitation of Hierarchical-PABO is scalability and ability to parallelize the approach. The goal of Hierarchical-PABO is predicting the Pareto region for a search space with reasonable ranges for the hyperparameters and with only few evaluations and we do not want to compete with all NAS-based approaches that search the entire search space with massive computational resource requirements. However, improving scalability of Hierarchical-PABO paves the way for incorporating the technique in different frameworks with multiple layers of optimization problems and hyperparameters.

For future work, we intend to fully integrate the Hierarchical-PABO approach into the TENNLab neuromorphic framework by Plank et al. (2018), so that it can seamlessly determine hyperparameters for the neuromorphic framework user. Within that framework, we also intend to apply this hyperparameter framework to other neuromorphic implementations that are supported and other applications, including a variety of control applications (like those described by Plank et al., 2019) and other classification tasks. We also plan to apply H-PABO to determine the hyperparameters for other spiking neural network training approaches, including reservoir computing algorithms, and back-propagation style approaches such as Whetstone (Severa et al., 2019) and SLAYER (Shrestha and Orchard, 2018). To further

accelerate the optimization approach, we plan to investigate an implementation of H-PABO for high-performance computers, such as Oak Ridge National Laboratory's Summit supercomputer.

DATA AVAILABILITY STATEMENT

The following datasets used in this study can be found at:

- Flower17: <http://www.robots.ox.ac.uk/~vgg/data/flowers/17/>
- CIFAR10: <https://www.cs.toronto.edu/~kriz/cifar.html>
- IRIS: <https://archive.ics.uci.edu/ml/datasets/Iris>
- Radio: <https://www.deepsig.io/datasets/>

All codes for Hierarchical-PABO as well as the simulation codes for pole balance and robotic navigation used in this work are available from the authors on request.

AUTHOR CONTRIBUTIONS

MP and KR defined the experimental setup and research experiments for the H-PABO approach for ANN domain, where the extra Bayesian optimizer block in the supervisor agent is off. MP, JM, and CS formulated the experimental setup and research experiments for the H-PABO approach for SNN domain. MP implemented H-PABO and conducted all of the experiments. RP and TP provided feedback and insight into the H-PABO approach for SNN. MP took the lead in writing the manuscript. All authors provided critical feedback and helped shape the research, analysis and manuscript.

FUNDING

This research was funded in part by Center for Brain Inspired Computing Enabling Autonomous Intelligence (C-BRIC), one

of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the National Science Foundation, Intel Corporation and Vannevar Bush Faculty Fellowship, U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725, and by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory. The funders were not involved in the study design,

collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fnins.2020.00667/full#supplementary-material>

REFERENCES

- Agrawal, S., and Goyal, N. (2013). "Thompson sampling for contextual bandits with linear payoffs," in *International Conference on Machine Learning* (Atlanta, GA), 127–135.
- Ankit, A., Hajj, I. E., Chalamalasetti, S. R., Ndu, G., Foltin, M., Williams, R. S., et al. (2019). "Puma: a programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI: ACM), 715–731. doi: 10.1145/3297858.3304049
- Baker, B., Gupta, O., Raskar, R., and Naik, N. (2017). Accelerating neural architecture search using performance prediction. *arXiv [Preprint]. arXiv:1705.10823*.
- Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). "Algorithms for hyperparameter optimization," in *Advances in Neural Information Processing Systems*, eds J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira and K. Q. Weinberger (Granda: Neural Information Processing Systems Foundation, Inc), 2546–2554.
- 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
- Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv [Preprint]. arXiv:1012.2599*.
- Bull, A. D. (2011). Convergence rates of efficient global optimization algorithms. *J. Mach. Learn. Res.* 12, 2879–2904. doi: 10.5555/1953048.2078198
- Cai, H., Zhu, L., and Han, S. (2018). Proxylessnas: direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.
- Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., and Rose, G. S. (2017). Memristive mixed-signal neuromorphic systems: energy-efficient learning at the circuit-level. *IEEE J. Emerg. Select. Top. Circ. Syst.* 8, 125–136. doi: 10.1109/JETCAS.2017.2777181
- Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., et al. (2019). "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 11398–11407. doi: 10.1109/CVPR.2019.01166
- Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 182–197. doi: 10.1109/4235.996017
- Dua, D., and Graff, C. (2017). *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences. Available online at: <http://archive.ics.uci.edu/ml> (accessed December 2019).
- Eggenberger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., et al. (2013). "Towards an empirical foundation for assessing Bayesian optimization of hyperparameters," in *NIPS workshop on Bayesian Optimization in Theory and Practice*, Vol. 10, 3.
- Esser, S., Merolla, P., Arthur, J., Cassidy, A., 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
- Esser, S. K., Appuswamy, R., Merolla, P., Arthur, J. V., and Modha, D. S. (2015). "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett (Montreal, QC: Neural Information Processing Systems Foundation, Inc), 1117–1125.
- Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2006). "Efficient non-linear control through neuroevolution," in *European Conference on Machine Learning* (Berlin: Springer), 654–662. doi: 10.1007/11871842_64
- 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*, eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett (Montreal, QC: Neural Information Processing Systems Foundation, Inc), 1135–1143.
- Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). "Predictive entropy search for efficient global optimization of black-box functions," in *Advances in Neural Information Processing Systems*, eds Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Montreal, QC: Neural Information Processing Systems Foundation, Inc), 918–926.
- 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*.
- Jin, J., Dundar, A., and Culurciello, E. (2014). Flattened convolutional neural networks for feedforward acceleration. *arXiv [Preprint]. arXiv: 1412.5474*.
- Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. *J. Global Optimizat.* 21, 345–383. doi: 10.1023/A:1012771025575
- Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. Global Optimizat.* 13, 455–492. doi: 10.1023/A:1008306431147
- Koo, M., Srinivasan, G., Shim, Y., and Roy, K. (2020). "SBSNN: stochastic-bits enabled binary spiking neural network with on-chip learning for energy efficient neuromorphic computing at the edge," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, ed A. James (IEEE), 1–10. doi: 10.1109/TCSI.2020.2979826
- Krizhevsky, A. (2009). *Learning Multiple Layers of Features From Tiny Images*. Technical report. Citeseer.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, eds F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Lake Tahoe, NV: Neural Information Processing Systems Foundation, Inc), 1097–1105.
- Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *J. Basic Eng.* 86, 97–106. doi: 10.1115/1.3653121
- Lai, T. L., and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.* 6, 4–22. doi: 10.1016/0196-8858(85)90002-8
- Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., et al. (2018). "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)* (Munich), 19–34. doi: 10.1007/978-3-030-01246-5_2
- Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.
- Marculescu, D., Stamoulis, D., and Cai, E. (2018). "Hardware-aware machine learning: modeling and optimization," in *Proceedings of the International Conference on Computer-Aided Design* (San Diego, CA: ACM), 137. doi: 10.1145/3240765.3243479

- Mitchell, J. P., Bruer, G., Dean, M. E., Plank, J. S., Rose, G. S., and Schuman, C. D. (2017). "Neon: neuromorphic control for autonomous robotic navigation," in *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)* (Ottawa, ON: IEEE), 136–142. doi: 10.1109/IRIS.2017.8250111
- Mitchell, J. P., Dean, M. E., Bruer, G. R., Plank, J. S., and Rose, G. S. (2018). "Danna 2: dynamic adaptive neural network arrays," in *Proceedings of the International Conference on Neuromorphic Systems* (Knoxville, TN: ACM), 10. doi: 10.1145/3229884.3229894
- Nilsback, M.-E., and Zisserman, A. (2006). "A visual vocabulary for flower classification," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2 (New York, NY: IEEE), 1447–1454. doi: 10.1109/CVPR.2006.42
- Panda, P., Sengupta, A., and Roy, K. (2016). "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Dresden: IEEE), 475–480. doi: 10.3850/9783981537079_0819
- Panda, P., Sengupta, A., and Roy, K. (2017). Energy-efficient and improved image recognition with conditional deep learning. *ACM J. Emerg. Technol. Comput. Syst.* 13:33. doi: 10.1145/3007192
- Parsa, M., Ankit, A., Ziabari, A., and Roy, K. (2019a). "PABO: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, (San Diego, CA), 1–8. doi: 10.1109/ICCAD45719.2019.8942046
- Parsa, M., Mitchell, J. P., Schuman, C. D., Patton, R. M., Potok, T. E., and Roy, K. (2019b). "Bayesian-based hyperparameter optimization for spiking neuromorphic systems," in *2019 IEEE International Conference on Big Data (Big Data)* (Los Angeles, CA: IEEE), 4472–4478. doi: 10.1109/BigData47090.2019.9006383
- Parsa, M., Panda, P., Sen, S., and Roy, K. (2017). "Staged inference using conditional deep learning for energy efficient real-time smart diagnosis," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, (Seogwipo: IEEE), 78–81. doi: 10.1109/EMBC.2017.8036767
- Parsa, M., Schuman, C. D., Date, P., Rose, D. C., Kay, B., Mitchell, J. P., et al. (2020). Hyperparameter optimization in binary communication networks for neuromorphic deployment. *arXiv [Preprint]*. arXiv:2005.04171.
- Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. (2018). Efficient neural architecture search via parameter sharing. *arXiv [Preprint]*. arXiv:1802.03268.
- Plank, J. S., Rizzo, C., Shahat, K., Bruer, G., Dixon, T., Goin, M., et al. (2019). "The TENNLab suite of LIDAR-based control applications for recurrent, spiking, neuromorphic systems," in *44th Annual GOMACTech Conference* (Albuquerque, NM).
- Plank, J. S., Rose, G. S., Dean, M. E., Schuman, C. D., and Cady, N. C. (2017). "A unified hardware/software co-design framework for neuromorphic computing devices and applications," in *2017 IEEE International Conference on Rebooting Computing (ICRC)* (Washington, DC: IEEE), 1–8. doi: 10.1109/ICRC.2017.8123655
- Plank, J. S., Schuman, C. D., Bruer, G., Dean, M. E., and Rose, G. S. (2018). The TENNlab exploratory neuromorphic computing framework. *IEEE Lett. Comput. Soc.* 1, 17–20. doi: 10.1109/LOCS.2018.2885976
- Rathi, N., Srinivasan, G., Panda, P., and Roy, K. (2020). Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv [Preprint]*. arXiv: 2005.01807.
- Reagen, B., Hernández-Lobato, J. M., Adolf, R., Gelbart, M., Whatmough, P., Wei, G.-Y., et al. (2017). "A case for efficient accelerator design space exploration via Bayesian optimization," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)* (Taipei: IEEE), 1–6. doi: 10.1109/ISLPED.2017.8009208
- Reagen, B., Whatmough, P., Adolf, R., Rama, S., Lee, H., Lee, S. K., et al. (2016). "Minerva: enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (Seoul: IEEE), 267–278. doi: 10.1109/ISCA.2016.32
- Reynolds, J. J., Plank, J. S., Schuman, C. D., Bruer, G. R., Disney, A. W., Dean, M. E., et al. (2018). "A comparison of neuromorphic classification tasks," in *Proceedings of the International Conference on Neuromorphic Systems* (Knoxville, TN: ACM), 12. doi: 10.1145/3229884.3229896
- Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Guettler, M., Hartel, A., et al. (2017). "Neuromorphic hardware in the loop: training a deep spiking network on the brainscales wafer-scale system," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 2227–2234. doi: 10.1109/IJCNN.2017.7966125
- Schuman, C. D., Plank, J. S., Bruer, G., and Anantharaj, J. (2019). "Non-traditional input encoding schemes for spiking neuromorphic systems," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest: IEEE), 1–10. doi: 10.1109/IJCNN.2019.8852139
- Schuman, C. D., Plank, J. S., Disney, A., and Reynolds, J. (2016). "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC: IEEE), 145–154. doi: 10.1109/IJCNN.2016.7727192
- Severa, W., Vineyard, C. M., Dellana, R., Verzi, S. J., and Aimone, J. B. (2019). Training deep neural networks for binary communication with the whetstone method. *Nat. Mach. Intell.* 1:86. doi: 10.1038/s42256-018-0015-y
- Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE* 104, 148–175. doi: 10.1109/JPROC.2015.2494218
- Shrestha, S., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnet (Montreal, QC: Neural Information Processing Systems Foundation, Inc), 1412–1421.
- Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Stamoulis, D., Cai, E., Juan, D.-C., and Marculescu, D. (2018). "Hyperpower: power- and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (Dresden: IEEE), 19–24. doi: 10.23919/DATE.2018.8341973
- Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., et al. (2019). "MNASNet: platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 2820–2828. doi: 10.1109/CVPR.2019.00293
- Wang, M., Liu, B., and Foroosh, H. (2017). "Factorized convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision* (Venice), 545–553. doi: 10.1109/ICCVW.2017.71
- Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, eds D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett (Barcelona: Neural Information Processing Systems Foundation, Inc), 2074–2082.
- Wieland, A. P. (1991). "Evolving neural network controllers for unstable systems," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, Vol. 2 (Seattle, WA: IEEE), 667–673. doi: 10.1109/IJCNN.1991.155416
- Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., et al. (2019). "FBNet: hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 10734–10742. doi: 10.1109/CVPR.2019.01099
- Xie, L., and Yuille, A. (2017). "Genetic CNN," in *Proceedings of the IEEE International Conference on Computer Vision* (Venice), 1379–1388. doi: 10.1109/ICCV.2017.154
- Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu, HI), 5687–5695. doi: 10.1109/CVPR.2017.643
- Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., et al. (2018). "NetAdapt: platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on*

- Computer Vision (ECCV)*, (Munich), 285–300. doi: 10.1007/978-3-030-01249-6_18
- Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). “ShuffleNet: an extremely efficient convolutional neural network for mobile devices” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 6848–6856. doi: 10.1109/CVPR.2018.00716
- Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 8697–8710. doi: 10.1109/CVPR.2018.00907

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.

Copyright © 2020 Parsa, Mitchell, Schuman, Patton, Potok and Roy. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Exploring Neuromodulation for Dynamic Learning

Anurag Daram^{1*}, Angel Yanguas-Gil² and Dhireesha Kudithipudi¹

¹ Neuromorphic AI Lab, University of Texas, San Antonio, TX, United States, ² Argonne National Labs, Lemont, IL, United States

OPEN ACCESS

Edited by:

Lei Deng,

University of California, Santa Barbara,
United States

Reviewed by:

Priyadarshini Panda,
Yale University, United States
Fangwen Yu,
Tsinghua University, China

*Correspondence:

Anurag Daram
anurag.daram@my.utsa.edu

Specialty section:

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

Received: 15 January 2020

Accepted: 11 August 2020

Published: 18 September 2020

Citation:

Daram A, Yanguas-Gil A and
Kudithipudi D (2020) Exploring
Neuromodulation for Dynamic
Learning. *Front. Neurosci.* 14:928.
doi: 10.3389/fnins.2020.00928

A continual learning system requires the ability to dynamically adapt and generalize to new tasks with access to only a few samples. In the central nervous system, across species, it is observed that continual and dynamic behavior in learning is an active result of a mechanism known as neuromodulation. Therefore, in this work, neuromodulatory plasticity is embedded with dynamic learning architectures as a first step toward realizing power and area efficient few shot learning systems. An inbuilt modulatory unit regulates learning based on the context and internal state of the system. This renders the system an ability to self modify its weights. In one of the proposed architectures, ModNet, a modulatory layer is introduced in a random projection framework. ModNet's learning capabilities are enhanced by integrating attention along with compartmentalized plasticity mechanisms. Moreover, to explore modulatory mechanisms in conjunction with backpropagation in deeper networks, a modulatory trace learning rule is introduced. The proposed learning rule, uses a time dependent trace to modify the synaptic connections as a function of ongoing states and activations. The trace itself is updated via simple plasticity rules thus reducing the demand on resources. The proposed ModNet and learning rules demonstrate the ability to learn from few samples, train quickly, and perform few-shot image classification in a computationally efficient manner. The simple ModNet and the compartmentalized ModNet architecture learn benchmark image classification tasks in just 2 epochs. The network with modulatory trace achieves an average accuracy of $98.8\% \pm 1.16$ on the omniglot dataset for five-way one-shot image classification task while requiring 20x fewer trainable parameters in comparison to other state of the art models.

Keywords: neuromodulation, ModNet, one-shot learning, dynamic learning, mushroom body output neurons (MBONs)

1. INTRODUCTION

Biological brains are capable of processing massive amounts of information for learning, retaining, and performing cognitive decision making. Moreover, the brains are endowed with the ability to learn continuously and adapt quickly to changes in the inputs or the environment in an energy efficient manner. The extraordinary computational capabilities of biological neural systems has motivated researchers to explore the structural and functional aspects of the brain, in order to build intelligent systems capable of solving complex tasks (Mead, 1990; Rumelhart et al., 1995; Kar, 2016; Hassabis et al., 2017). Using the brain as a source of inspiration, researchers have successfully demonstrated networks with the ability to solve complex learning tasks. For example,

convolutional neural networks (Lawrence et al., 1997; Huang et al., 2017) have demonstrated remarkable performance for image recognition tasks, recurrent neural networks (Williams and Zipser, 1989; Gers et al., 2002; Greff et al., 2017; Sak and Senior, 2018) have been able to perform classification, prediction, and anomaly detection on temporal tasks, and few reinforcement learning based systems (Sutton and Barto, 1998; Silver et al., 2016) are able to learn complex cognitive tasks.

While such state-of-the-art networks perform well on narrow sets of well-defined tasks, they are generally not very good at generalizing, and they are not able to learn from few samples (Bengio et al., 2015; Rosenfeld et al., 2018). To address these issues, biological brains have mechanisms that dynamically adjust its own parameters for learning in new environments. These mechanisms play a key role in reacting and responding to stimulus based on context in a quick and efficient way. It has been observed in many species—from insects to humans—that, in addition to synaptic plasticity, neuromodulation plays a key role in the facilitation of learning (Decker and McGaugh, 1991). Brains also use neuromodulation to modify neural connectivity in response to inputs and internal states. Neuromodulation is the physiological process by which a given neuron uses one or more neurotransmitters to regulate a population of neurons (Katz and Edwards, 1999). This contrasts with classical synaptic transmission, in which one presynaptic neuron directly influences a single postsynaptic partner. Neuromodulators secreted by a small group of neurons diffuse through large areas of the nervous system thus affecting multiple neurons. Reports have shown that neuromodulation affects synaptic plasticity, neural wiring, and attention (Katz, 1999; Doya, 2002).

In this work, we develop computationally efficient dynamic learning systems inspired from neuromodulatory mechanisms in the brain wherein a modulatory unit regulates the learning according to the context and the internal state of the system. Here, the internal state of the system refers to the activations of the neurons in response to the current input. When we refer to dynamic learning, we focus on the capability of associative learning; where the system learns to discriminate its input based on a context, which can either be internal to the system or triggered by an external input, such as a reinforcement or a modulatory signal. In addition to implementing dynamic learning capabilities, our architecture needs an attention component responsible for meta-learning; its main function is to evaluate when, what, and how much to learn based on the context. Thus one approach toward solving this problem is by incorporating the heterosynaptic (neuromodulatory) mechanisms in conventional neural networks.

Some researchers have incorporated the concept of neuromodulated plasticity into network models for solving tasks in dynamic reward-based scenarios. Soltoggio et al. (2008) proposed an architecture where they introduced the concept of modulatory neurons that enhances or dampens the neural plasticity of the target neurons to boost the memory and learning capabilities. The concept of gated plasticity in Soltoggio et al. (2008) enabled dynamic targeted update of synapses in the network, thus leading to more efficient learning. The work in Miconi et al. (2020), demonstrates that adding neuromodulatory

plasticity mechanisms trained using gradient descent exhibit superior performance on reinforcement learning and non-trivial supervised learning tasks like few shot learning, pattern memorization, and image reconstruction. They also explain that self-modifying capabilities in the brain play an important role in learning and adaptation. This work shows that incorporating these learning mechanisms along with an architecture inspired from insects enables learning dynamically and from few samples in a computationally efficient fashion.

The key contributions of this work are:

- Incorporating architectural and functional methods inspired from the insect brain to enable neuromodulatory interactions in conventional neural networks.
- Adaptive local learning rules with built-in attention mechanisms that endow the networks with the capability to learn from few-samples.
- A compartmentalized network architecture akin to the mushroom body in the drosophila to process the information in a scalable and resource efficient way.
- A modified modulatory trace learning rule capable of learning and efficiently processing inputs from the internal state of the system.

2. RELATED WORK

2.1. Neuromodulated Plasticity in Neural Networks

In the brain, the neurons communicate with each other by releasing neurotransmitters when the axon potential of the neuron reaches a synapse. Depending on the type of the neurotransmitter, the receiving neuron can be in either excitatory or inhibitory state. Neurotransmitters can sometimes cause an electrical signal to be transmitted down the cell (excitatory), whereas in other cases, the neurotransmitter can actually block the signal from continuing, thereby preventing the message from being carried on (inhibitory). Some of the neurotransmitters that have spatially distributed and temporally extended effects on the recipient neurons and circuits, are called Neuromodulators (Katz and Edwards, 1999). The best examples of neuromodulators are dopamine, serotonin, noradrenaline (also called as norepinephrine) and acetylcholine. Doya (2002) hypothesized the role of different neuromodulators in the context of reinforcement learning in the brain. His hypothesis was as followed: Dopamine acts as the global control and learning signal for the network for predicting rewards and reinforcement of actions. Serotonin modulates the balance between the short-term and long-term prediction of rewards. Similarly, noradrenaline modulates the attention mechanism in the network in the sense that it controls the balance between wide exploration and focused execution. Acetylcholine handles the memory and controls memory storage and renewal of memory. It modulates the learning wherein, based on acetylcholine release, learning new tasks and rate of forgetting of previously learned tasks is handled. Following that, there have been several other hypotheses on similar lines (Bargmann, 2012; Pedrosa and Clopath, 2017; Shine et al., 2019)

regarding the functional role of the neuromodulators in the brain. Taking inspiration from this, several researchers have incorporated neuromodulation in deep learning frameworks. Kondo (2007) proposed a diffusion-reaction based model using neuromodulation. The neuromodulators via those actions, regulate the synaptic connectivity and strength. This mechanism was able to demonstrate online learning capabilities for mobile robotic control. The concept of neuromodulated spike timing dependent plasticity in spiking neural networks is introduced by Frémaux and Gerstner (2016). These gated plasticity based learning rules show how neuromodulatory signals interact with the neural activity to bias learning and behavior, and respond to novelty. Kolouri et al. (2019) proposes an attention based selective plasticity approach that is based on cholinergic modulation in the brain to address catastrophic forgetting. The central idea in most of the previous works portrays the ability of neuromodulators to impact plasticity predominantly through gating of plasticity and up-regulation of neuronal activity. These features or effects of neuromodulators are observed across multiple species not only including mammals and reptiles, but also insects.

There is an active research aiming to understand how smaller brains can be highly capable of learning and cognition (Montgomer et al., 2018). Despite having brains that are a million times smaller, insects are able to exhibit almost half the distinct cognitive behaviors as that of certain mammals like dolphins (Changizi, 2013; Theobald, 2014) (59 for honeybees compared to 123 for dolphins). For example, bees build honeycombs and operate in swarms via symbolic communication, wasps exhibit chemical communication, termite colonies perform climate control, etc. The neural circuitry found in insect brains is able to exhibit complex cognitive behaviors similar to mammals albeit with a lower resolution and reduced information processing (Lihoreau et al., 2012). Moreover, cognitive ability does not necessarily result from greater numbers of neurons but rather it is the new links between different bundles of neurons that lead to tangible changes in behavior (Chittka and Niven, 2009). Yanguas-Gil et al. (2019) shows that architectures inspired from insect brain are capable of exhibiting context-dependent processing and learning. Therefore, models based on small brains can still offer a good baseline of intelligent tasks in a resource and power efficient manner.

2.2. Few Shot Learning

Several real-world application domains like healthcare, robotics, etc., operate on irregular and sparse datasets. To address this issue, few shot learning is becoming a prominent area of research. However, learning and adapting from few examples is very challenging. The conventional approaches for image classification involving convolutional neural networks trained using backpropagation are unable to offer a satisfactory solution for learning new concepts rapidly from little data. Hence, there have been few works that were particularly inclined toward solving this problem and have been able to achieve good performance on few shot learning tasks. The Siamese network model (Koch et al., 2015), tries to approach the problem of few shot learning by giving the model two samples and then

training it to guess whether the two samples belong to the same category or not. Another approach to the few shot learning task is specified in Matching Networks (Vinyals et al., 2016). Matching Nets use novel attention mechanisms and embedding functions to enable rapid learning and train the network by showing only few examples per class. They train the network by randomly selecting k labeled examples from N classes that haven't previously been trained upon. The task is then to classify a batch of unlabeled examples into one of these N classes. The model proposed in Mishra et al. (2017) currently achieves state of the art performance for few shot learning tasks. The authors introduced temporal convolution (TC) and causal attention layers in the network, wherein the TC layers provides the context over which the attention layers operate.

Apart from the prior specified techniques, researchers have proposed meta-learning based techniques. The work proposed by Santoro et al. (2016) uses a novel sophisticated memory based system. It uses Long Short-Term Memories (LSTMs) as a memory controller that interfaces with the input and outputs through complex memory accesses. The network learns a general strategy for the types of representations it should place into memory and how it should later use these representations for predictions. Recently, Finn et al. (2017) proposed Model-Agnostic Meta Learning for fast adaptation of Deep networks, that introduces a meta learning algorithm that can be trained with any model with gradient descent and can be used to solve a variety of problems like classification, regression and reinforcement learning. Researchers (Doya, 2002; Soltoggio et al., 2008; Miconi et al., 2020) have studied the role of neuromodulatory and heterosynaptic update mechanisms for endowing networks with meta-learning capabilities. In this work, the authors use plasticity based rules to encode the context and drive the update of parameters in the network in conjunction with backpropagation.

3. MODULATION INSPIRED LEARNING METHODS

In the proposed work, the efficacy of adding neuromodulation to the neural networks is observed. The first approach couples synaptic local learning rules with error driven modulation to enable learning on the edge. The second approach incorporates neuromodulation in conjunction with backpropagation wherein a context driven modulatory trace regulates the short term plasticity of the connections.

3.1. ModNet

The proposed architecture, Modulatory Network (ModNet), Daram et al. (2019) derives its inspiration from the mushroom body in the insects and the learning mechanism is inspired from the neuromodulatory mechanisms in the brain. Neuromodulators closely affect synaptic plasticity, neural wiring and the mechanisms of long term potentiation (LTP) and long-term depression (LTD). The realization that Hebbian learning is not the only way that synapses are modified (Cooper, 2005) has led to growing interest in neuromodulation. Studies

on mollusks and insects (Carew et al., 1981; Roberts and Glanzman, 2003) have shown that in addition to Hebbian learning, neuromodulatory mechanisms are also involved with associative learning and synaptic changes.

The learning rule proposed in our architecture derives from the aforementioned heterosynaptic mechanisms and uses the concept of Hebbian plasticity for the synaptic weight update. The pre-synaptic and post-synaptic neurons determine the polarity of change in the connection while the modulatory neurons regulate the rate at which the weight is updated.

The mushroom body output neurons (MBONs) in the *Drosophila* play a key role in discriminating between stimuli, learning their predictive value and further using that information to modify their behavior (Aso et al., 2014). Additionally, dopaminergic modulation alters the balance within the MBON network for those stimuli. The input layer in the ModNet corresponds to the antennal lobe projection neurons (sensory stimuli). These antennal lobe neurons are sparsely represented in the Kenyon cells and a similar property is used in ModNet, as shown in **Figure 1** (Daram et al., 2019), the inputs are randomly projected into a sparse hidden space. Sparsity ensures greater feature separability and distinctive representation of the inputs. The Kenyon cells then converge into multiple MBONs and the plasticity of those connections is regulated by neuromodulation based on stimuli. Similarly, in ModNet, the hidden layer is fully connected to the output layer and the plasticity of those connections is regulated by a modulatory layer. This modulatory layer takes as input the error calculated at the output layer and uses it to regulate the plasticity of the hidden-to-output layer weights.

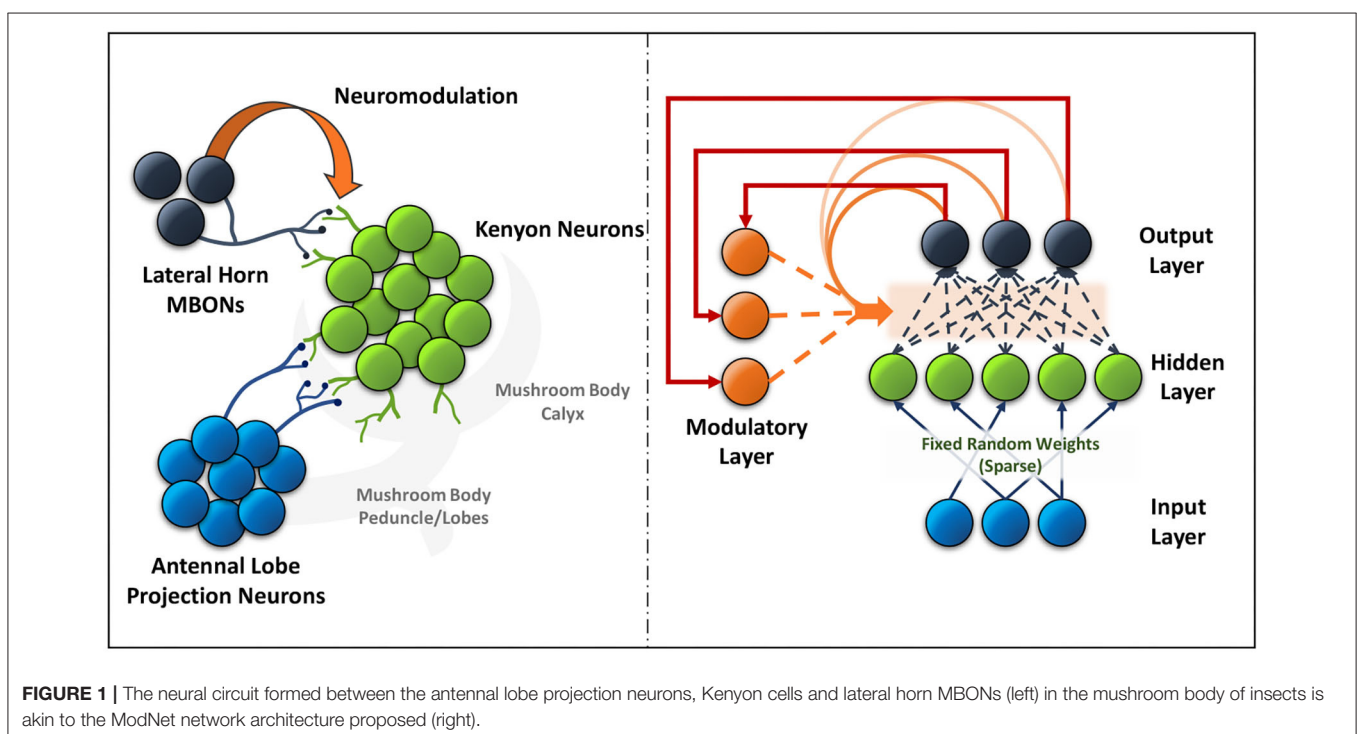
The network consists of two units, the processing unit which is responsible for learning the features and the distinctive

representations, and the neuromodulatory layer which is responsible for learning the context. In the processing unit, the input features are lifted onto a higher dimensional hidden layer which extract the spatial features. These features are processed through sigmoid activation function at the hidden layer and are learned at the output layer. The learning error from the output neurons, with respect to a one-hot encoded label as input, are passed as inputs to the modulatory layer. The modulatory neurons compensate the error by updating the trainable weights from the hidden to output layer neurons. During the training phase, the output neurons have a set of two activations each, namely the standard activation and the modulatory activation. The standard activations are computed as sum of the products of hidden-neuron activations and the hidden-to-output layer weights. The modulatory activations are computed as sums of products of the modulatory inputs to the modulatory weights. The standard and the modulatory activations are calculated as shown in (1) and (2).

$$A_i = \text{Sigmoid} \left(\sum w_{ij} x_j \right) \quad (1)$$

$$M_i = \sum w'_{ij} x'_j \quad (2)$$

where w_{ij} corresponds to the hidden to output layer weights from the i^{th} neuron in the output layer to the j^{th} neuron in the hidden layer, and x_j corresponds to the hidden layer activations. w'_{ij} corresponds to modulatory weights from the i^{th} neuron in the output layer to the j^{th} neuron in modulatory layer and x'_j correspond to inputs to modulatory neurons (error computed at the output layer), respectively. Once the activations



are calculated, the standard and modulatory weights are updated as shown in Equations (3) and (6), respectively.

$$\Delta w_{ij} = \text{Sigmoid}\left(\frac{M_i}{n_{ij}}\right) \times \delta_{ij} \quad (3)$$

The weight update equation has two components with the first part being the magnitude component and δ_{ij} being the plasticity and direction term. In (3), n_{ij} is a scaling parameter that is tuned while training. The plasticity term δ_{ij} is realized according to Equation (4),

$$\delta_{ij} = \eta_{ij} (\beta_1 x_i x_j + \beta_2 (x_j - x_i) + \beta_3) \quad (4)$$

where η_{ij} is the adaptive learning parameter that is updated while training, x_i and x_j are the pre and post-synaptic activations, and β_1 , β_2 , and β_3 are the tunable parameters for the network. The weight update equation has a correlation term β_1 , a difference term β_2 and a constant term β_3 as a bias. The constant term allows for update of the synapse even in absence of pre or post synaptic activation. The polarity of η_{ij} is changed based on the difference between the activations and the polarity of the connecting weight. Hence in this learning rule, the modulatory component regulates the magnitude of the rate of weight change and the plasticity component determines the sign or direction of the weight update for the given connection. This term selects and strengthens the set of connections contributing toward learning a particular task. The adaptive learning rate is updated according to Equation (5).

$$\eta_{ij} = \eta_{in} \frac{e_i}{x_i}, \quad (5)$$

where η_{in} corresponds to the initial value of the learning rate and e_i and x_i correspond to the error and the activation at the observed output neuron. In the case when e_i or x_i are 0, then the η_{ij} is set to η_{in} . Having the output activation as a divisive factor enables a more optimized rate of change in learning rate based on how far it is from correctly learning the associations. The same equation is also used for updating the modulatory weights, with the error and the activation terms switching positions. This mechanism is similar to attention mechanism in neural networks. The magnitude term that depends on the modulatory interactions, is also affected by the division term which changes the dynamic range of the sigmoid by flattening the curve. The modulatory weights are updated based on Equation (6),

$$\Delta w'_{ij} = \eta'_{ij}(\text{scale}) \quad (6)$$

where η'_{ij} is an adaptive learning parameter and scale is a tunable magnitude parameter. The sign and magnitude of η'_{ij} is updated as a function of the network response and the output activations. The sign of the learning parameter is directly correlated with the error and the magnitude is increased or decreased based on the value of output activation. The learning rule proposed in ModNet (Equations 3 and 4) consists of Hebbian update coupled with a modulatory regularizer wherein the rate of weight change is either enhanced or dampened with respect to the hidden layer

neurons' contribution toward learning. The proposed learning rule enables dynamic learning in the system with exposure to only few samples. But having a sparse hidden layer can lead to many redundant and unused neurons and synapses. Thus, to make the algorithm more efficient, a dynamic attention based mechanism is proposed.

3.2. Region Based Attention Mechanism

ModNet essentially shows a baseline network incorporating hetero-synaptic interactions in neural networks. But the network can be designed to be more dynamic to solve complex tasks in a further efficient fashion. Thus a mechanism to add attention in the hidden layer is introduced in this section. The algorithm implements an attention mechanism that distinctly selects populations of excitatory or active neurons while inhibiting and filtering the less active ones. This separation enables efficient distribution of information within the network. Thus, the hidden layer in the ModNet is divided into regions based on a scaling factor α to perform selective filtering and inhibition. This mechanism is adopted while training. The activities of the neurons in different regions are measured and a normalized average of neuronal activities in every region is computed. The activity factors of the regions are computed periodically for every 100 samples. If a_{ik} is the activation value of a neuron in k_{th} region, then the average activity factor of the region is given by Equation (7).

$$A_k = \frac{\frac{1}{a_{max}} \sum a_{ik}}{N} \quad (7)$$

where a_{max} is the maximum activation value in the given region and N represents the number of hidden neurons in the region. Based on the activity factor, the weights of every region are further updated according to the Equation (8).

$$w'_{ik} = w_{ik} - A_k \Delta w_{ik}, \quad (8)$$

where A_k controls the rate of change in synaptic strength and boosts the strength of the connections for the more active regions. If the activity (A_k) is less than a threshold value δ , then the weights of the neurons in those regions are further updated according to the Equation (9).

$$w'_{ik} = \begin{cases} w_{ik} - |A_k \Delta w_{ik}| & \text{when } w_{ik} > 0 \\ w_{ik} + |A_k \Delta w_{ik}| & \text{when } w_{ik} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

As shown in Equation (9), in the regions with lower average activity or smaller activity factor, the connections of the neurons in those regions are inhibited and the weights are made to converge to 0 thereby making those regions sparser. Thus, the region based attention mechanism is able to determine the active and inactive regions during training and is dynamically able to drop out connections and neurons while retaining the performance. This mechanism further enables introducing a more complex and mushroom body inspired architectural formulation known as compartmentalization.

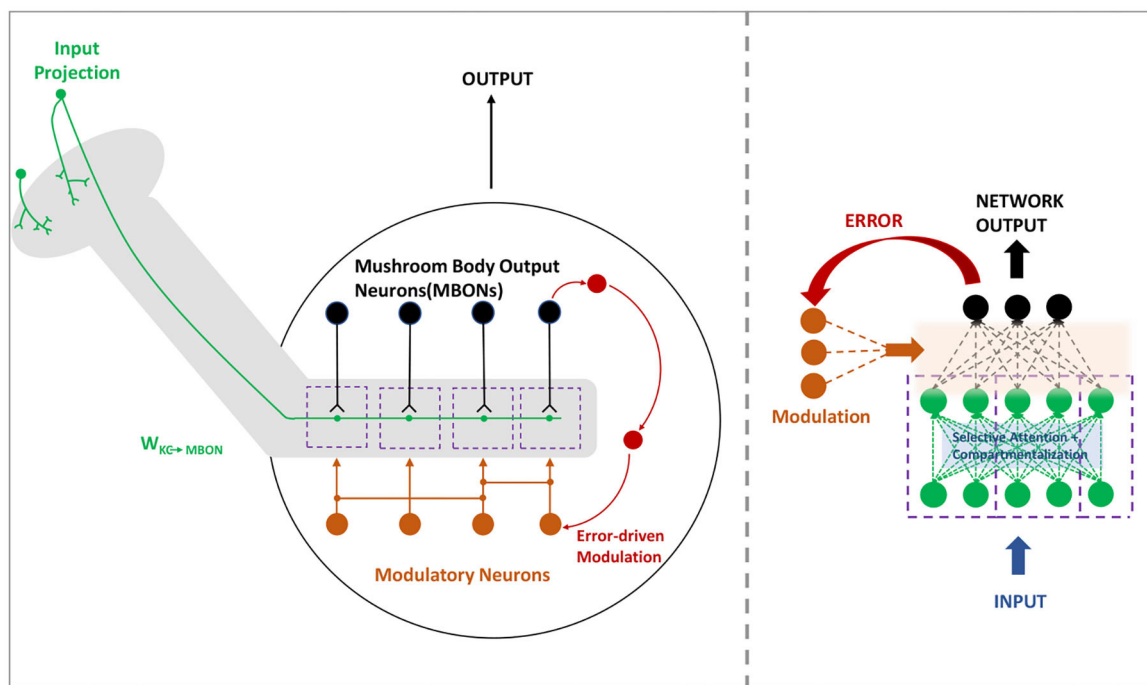


FIGURE 2 | The mushroom body (left) in insects are organized in a compartmentalized fashion to encode and bias the behavior. Compartmentalized ModNet architecture (right) incorporates this design aspect to perform selective filtering and resource efficient processing of information.

3.3. Compartmentalized ModNet

In the mushroom body, the MBONs are able to encode the valence and bias the behavior in a highly compartmentalized fashion (Aso et al., 2014). Every compartment in the mushroom body lobe acts as an independent valuation module with different and changing modular functions. Thus this idea is translated to a deeper version of the region based ModNet, with each region actively learning and acting as independent modules in the network. The core idea behind compartmentalization is to generate a modular network capable of adapting to different tasks with different regions learning to respond to certain tasks with only few data samples. **Figure 2** shows the neural architecture of the compartmentalized organization in the mushroom body output neurons in insects.

An additional hidden layer is added to the ModNet architecture. The layers are split into explicit multiple compartments competing to learn the features and eventually gating the less active ones. The net activity factor of each compartment is measured and the connectivity is updated based on the rule presented in Equations (8) and (9). The net activity factor for each compartment is measured as a factor of the activation values of the population of neurons in the respective compartment. The local connections between the neurons in the compartments are trained via local plasticity rules. Unlike gradient descent where the correlations between hierarchical layers are retained by gradients, the problem with having multiple rules across layers causes an issue of uncorrelated knowledge transfer. Hence, an unsupervised covariance based learning rule as shown in Equation (10), is utilized which actively

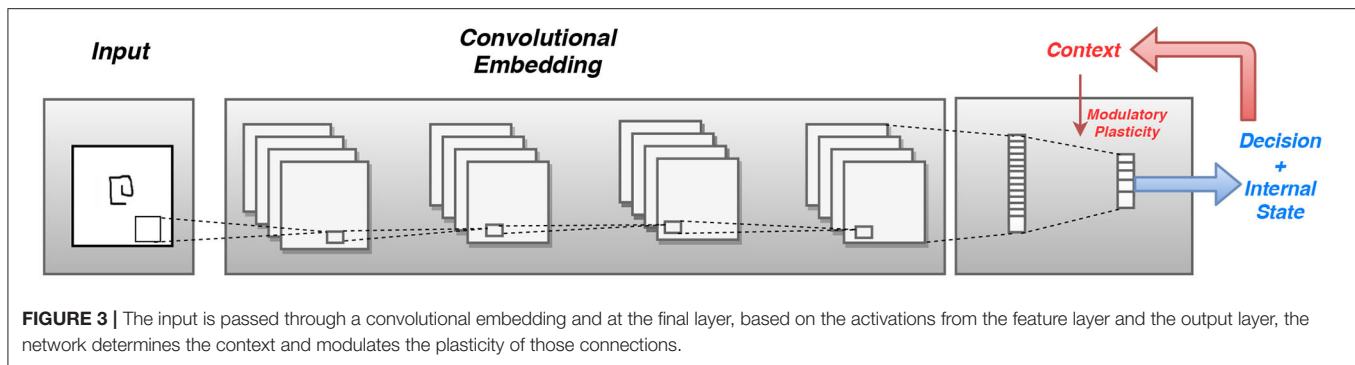
updates the synapses based on correlations between the pre and post synaptic neurons.

$$\Delta w_{ij} = \eta'[(x_{pre} - s1)(x_{post} - s2) \times w] \quad (10)$$

In Equation (10), $s1$ and $s2$ are hyperparameters that define the threshold for activations for the pre and post synaptic neurons to either strengthen or weaken the connection. Moreover, to alleviate the problem of oscillation of neuronal activity due to multiple learning rules, the covariance based learning rule is operated on a slower timescale, thus being updated after every N inputs wherein N can be set as a hyperparameter. The preliminary results demonstrate superior performance than the ModNet architecture on the Fashion-MNIST dataset.

3.4. Modulatory Plasticity in Deep Networks

ModNet architecture represents one way of introducing modulatory dynamics to a network and training via modulatory and plastic learning rules. However, the problem with ModNet lies in that being a shallow network, the capabilities of ModNet is limited to solving simpler classification tasks. However, the use of a gradient descent mechanism like backpropagation in neural networks has shown to achieve spectacular results in solving complex tasks. Thus, it would be interesting to add modulatory plasticity to these non-plastic backpropagation based neural networks. This way, not only will the weights be optimized while training via backpropagation but, also the plasticity in each connection is updated via neuromodulation. To test the



advantage of having neuromodulated plasticity in the context of lifelong learning, the task of performing efficient one-shot and few-shot learning is considered.

Figure 3 shows the concept of incorporating modulatory dynamics into a CNN for solving the task of few shot learning. Thus, to train a modulatory network with backpropagation, a formulation is considered in which each connection has a standard component and a modulatory component. This formulation is inspired from the work in Miconi et al. (2018), wherein the author considers the network to have a fixed and a plastic component for training. However, the plasticity component in Miconi et al. (2018), does not account for the full internal state of the network and thereby does not encode the global context while updating weights in the network. Hence, the connection between any neuron i in layer l and another neuron j in layer $l-1$ will have a regular connecting weight w_{ij} which constitutes the standard component, and a modulatory trace Mod_{ij} which changes based on the current inputs and outputs to the network. Thus the total weight of the connection between any two neurons is given by the sum of the regular weight and the modulatory trace. This is specified in Equation (11).

$$W_{tot} = w_{ij} + \delta_{ij}Mod_{ij}, \quad (11)$$

where δ_{ij} is the modulatory learning factor that can be either constant for all the connections or different for each. The role of the modulatory trace is to perform heterosynaptic weight update of the connections in the network. Hence the output activation is computed as a sum of products passed through a non-linear activation function of the input activations and the W_{tot} as represented in Equation (12).

$$x_{out} = \sigma \left(\sum_{inputs} [w_{ij} + \delta_{ij}Mod_{ij}]x_{in} \right), \quad (12)$$

where x_{out} corresponds to the output activation value and x_{in} corresponds to the input activations feeding into the output neuron. Here, σ corresponds to the non-linear activation function and the inputs correspond to the input activations from the previous layer.

The modulatory trace is a time dependent quantity for the connections in the network. The trace is updated based on

the input and the output activations and a modulatory context which is responsible for handling the short term memory in the network. The modulatory trace is computed according to the Equation (13).

$$Mod_{ij}(t) = Mod_{ij}(t-1) + \alpha_{ij}[x_{out}(x_{in} - x_{out}Mod_{ij}(t-1))], \quad (13)$$

where $Mod_{ij}(t)$ is the currently computed trace value and $Mod_{ij}(t-1)$ corresponds to the initial trace value or the trace value for the previous iteration. The modulatory trace is initialized to zero at the beginning of each epoch or episode. The parameters w_{ij} and δ_{ij} are trained across all the training epochs and episodes. These parameters are updated and optimized using backpropagation during the training process. α_{ij} is defined by the Equation (14).

$$\alpha_{ij} = \gamma \sigma' \left(\left[\sum_{outputs} x_{out} \right] / n \right), \quad (14)$$

where σ' is the non-linear activation function, sigmoid in this case and n is the number of output neurons in the layer and γ is a hyperparameter to regulate the rate of update. The outputs correspond to all the output activations. The α term appears as the modulatory context term which evaluates the network response to encode the global context to the trace.

The parameter α determines the speed at which new information is incorporated into the trace and the plastic component of the network while δ determines the magnitude of effect of the trace on the respective connection. The Mod_{ij} term or the trace is accumulated over time but gated by the modulatory context term and the output activation. The Mod_{ij} term is an episodic quantity, in the sense that it is reset to zero after every training episode. The modulatory context term and the weights are lifetime quantities as they are updated throughout the training procedure. The modulatory trace in the output layer evaluates the internal state of the system allowing for stable memories, thus enabling the connections to learn the associations between the inputs during the episode. This corresponds to the short term effect of the trace while training during the episode. The modulatory learning factor regulates the long term effect of the trace on the connections in the sense that, being a lifetime parameter, the context term encodes how much, the heterosynaptic update is required for the given task or episode.

TABLE 1 | Experimental setup and hyperparameters for evaluating the ModNet architecture.

Network features	ModNet			
Datasets	MNIST		Fashion-MNIST	
Network size	100 × 1,000 × 10		784 × 1,000 × 10	
Hyperparameters	β_1	β_2	β_3	η
	0.1	0.2	0.001	0.001-0.01
Training epochs/Episodes	2			
Runs	10			

The modulatory context term reinforces the connectivity and quickly updates the weights in the final layer to adapt to the newer inputs and thus differentiate between the incoming input features from n selected classes in a n -way k -shot learning scenario. This learning mechanism is thus able to perform competitively on the few shot learning task on the Omniglot dataset. This can be attributed to the long term and short term effects of the heterosynaptic mechanisms on the network that are responsible for understanding the context and regulating the response to the encoded context.

4. RESULTS AND ANALYSIS

In the proposed work, the ModNet architecture and the Modulatory trace learning method are evaluated and tested on different datasets based on the applications. The ModNet architecture is tested on the MNIST (LeCun et al., 1998) and the Fashion-MNIST (Xiao et al., 2017) image recognition benchmarks for classification, and the Modulatory trace learning method is verified on the Omniglot dataset (Lake et al., 2015) for few-shot classification task. In this section, the proposed network architectures are analyzed for performance and efficiency.

4.1. Image Classification Tasks

The ModNet architecture coupled with the learning and architectural mechanisms proposed in Sections 3.1-3.3 are evaluated for resource efficient dynamic learning on image classification benchmarks.

4.1.1. Benchmarks and Experiment Setup

Two spatial datasets are used to study the capability of ModNet in learning from few samples. Both the datasets consist of 60,000 training and 10,000 testing images. The MNIST dataset images are normalized and reshaped from 28×28 to 10×10 . The images from the Fashion-MNIST dataset are only normalized and not reshaped to 10×10 to prevent loss of useful features.

Table 1 shows the network setup for evaluating the ModNet architecture. The ModNet architecture configuration is set to 100 (input) × 1,000 (hidden) × 10 when testing on the MNIST dataset. The input layer size is updated to 784 while testing on the Fashion-MNIST dataset. In the case of Compartmentalized-ModNet, an additional hidden layer of size 1,000 is added to the ModNet network and analyzed.

4.1.2. ModNet Performance

As observed in Figure 4, the network is able to attain a test accuracy of ~91% while training for just 2 epochs on the MNIST dataset and ~81% on the Fashion-MNIST dataset. Figure 4A shows the training performance of ModNet with respect to the number of samples shown. The network is compared to a similar shallow random projection based network, called the Extreme Learning Machine (ELM) (Huang et al., 2004) which is unable to learn quickly as ModNet. This is a result of the effect of modulatory activations on weights in the network which initially try to reward and penalize neurons at a much higher rate as compared to the later stages when learning begins to saturate. Figure 4B shows how higher dimensionality leads to better performance, which correlates well with the biological counterpart in which the mushroom body projection neurons are lifted in the Kenyon cells. Higher dimensionality in the hidden layer results in a greater feature separation amongst the inputs.

On the contrary, the network performance does not improve significantly after increasing the size of the hidden layer past 2,500 neurons. However, the larger network performs better if it is trained longer. This is a result of the inability of the shallow network to capture certain distinctive features that could be realized by increasing the depth of the model. Therefore, the compartmentalized topology of ModNet and the attention mechanism are evaluated to address the depth of the network and efficient processing for shallower networks, respectively.

4.1.3. Exploiting Sparsity for Efficient Processing

To test the efficacy of compartmentalization with attention on ModNet, the network is re-evaluated on the Fashion-MNIST dataset. For the compartmentalized network, another hidden layer is added with the network configuration being 784 (input) × 1,000 (hidden1) × 1,000 (hidden2) × 10 and selective attention in the compartments in the hidden layers. The network is able to classify with an accuracy of ~91% on the Fashion-MNIST model. Figure 5A shows how selective filtering using attention with compartmentalization and gating allows efficient processing. Turning off least responding compartments and keeping only active compartments while training the network thus ensures greater resource and power efficiency.

Figure 5B shows the effect of region based attention mechanism and sparse initialization on the performance for the shallow ModNet architecture. We can observe similar performance for a 20–30% sparsely gated network in comparison to the fully connected network. However, other than sparse initialization, the attention mechanism also induces sparsity into the network. To find out the optimal sparsity for the network to perform comparably to the fully connected counterpart, and realize the effect of the aforementioned mechanisms, further analysis is performed as shown in Figures 6, 7.

Figure 6 shows the percentage of connections pruned while training using attention and compartmentalized topologies. Compartmentalization induces greater sparsity as the network has more connections, and a modular topology leads to compartments shutting off after falling below minimum activity threshold. Moreover, to observe the behavior of these mechanisms for already sparse or gated networks, the network

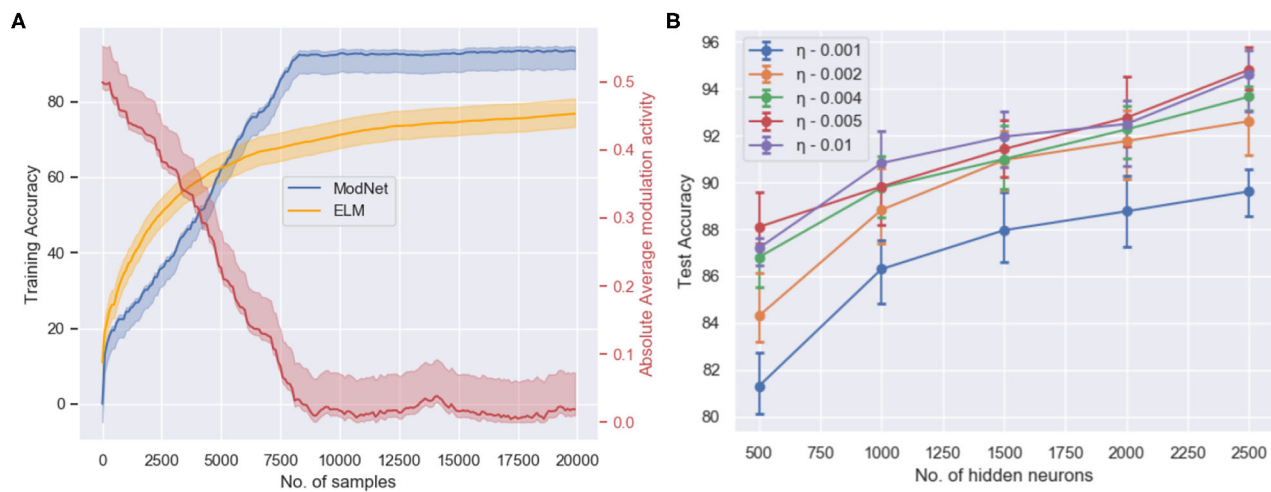


FIGURE 4 | (A) The training accuracy (blue) on the MNIST dataset with respect to the number of trained samples and the change in absolute modulatory activation values (red) for a hidden layer of size 1,000. **(B)** Test accuracy with respect to the number of hidden layer neurons and the initialized value of the adaptive learning parameter when trained for 2 epochs on the MNIST dataset for the ModNet architecture.

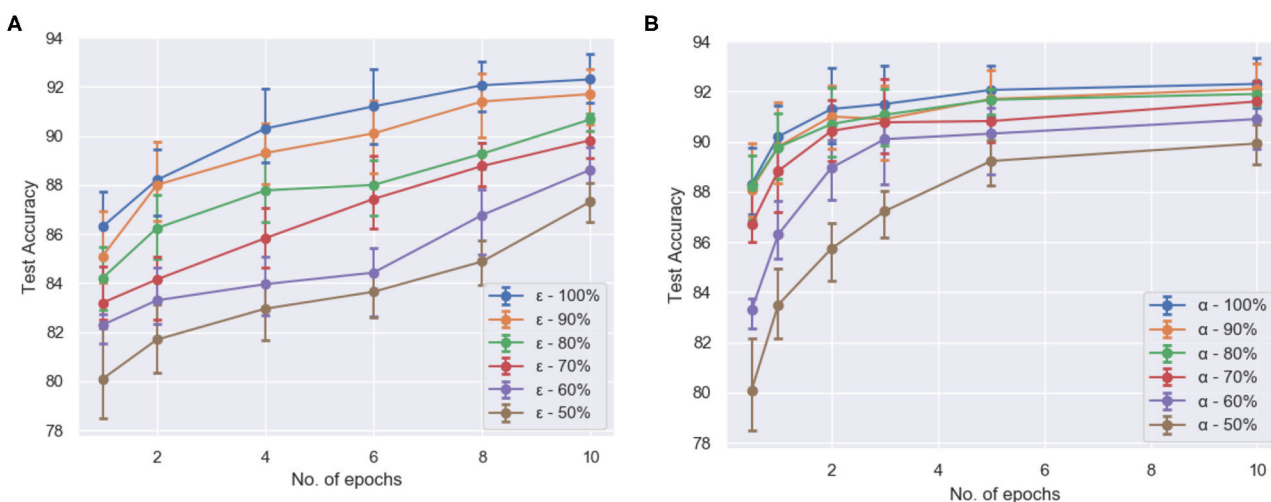


FIGURE 5 | The test accuracy (averaged over 5 runs each) **(A)** of compartmentalized ModNet with respect to the number of epochs and ϵ (initial % of randomly selected active neurons) for performing classification on the Fashion-MNIST dataset and **(B)** of ModNet with attention (without compartmentalization) with respect to the number of epochs and the initial percentage of the active regions (α) on the MNIST dataset.

sparsity after one epoch for these networks is observed. Based on the results in **Figure 7**, the optimal sparsity point is observed to be $\approx 30\%$ for compartmentalized ModNet and $\approx 40\%$ for region based attention mechanism. This demonstrates the efficient adaptation aspect of the proposed learning topologies. The learning mechanisms adapt to the different network initializations and try to converge to the optimal sparsity point necessary for performing the task without degrading the performance. However, increased sparsity also affects the network performance in other aspects. Thus, an ablation study is conducted to observe the problems and advantages of each of the proposed mechanisms.

4.1.4. Ablation Study

Table 2 presents the ablation study to demonstrate the efficacy of incorporating different mechanisms in addressing learning from few samples. The results shown in the table are averaged over 10 runs and the number of samples to reach initial convergence are an approximated average of the values over the runs. Coupling attention with the modulatory learning rule ensures reaching convergence quickly while saving on resources by increasing sparsity in the network. As discussed in section 4.1.3, compartmentalization is able to make the network more sparse, but it takes longer to converge as compared to the shallow counterparts. This is also attributed to increased depth and

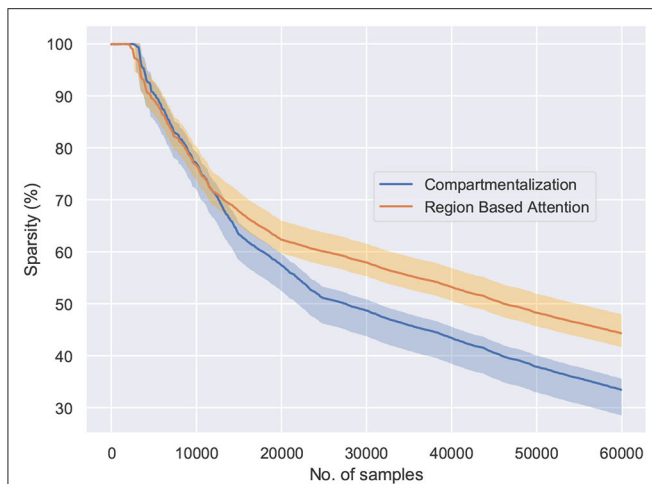


FIGURE 6 | The behavior of the network connectivity with respect to the two plasticity topologies when trained on the MNIST dataset. Compartmentalization induces greater sparsity in the network as a result of increased parameters and finds the optimal sparsity configuration for the given task.

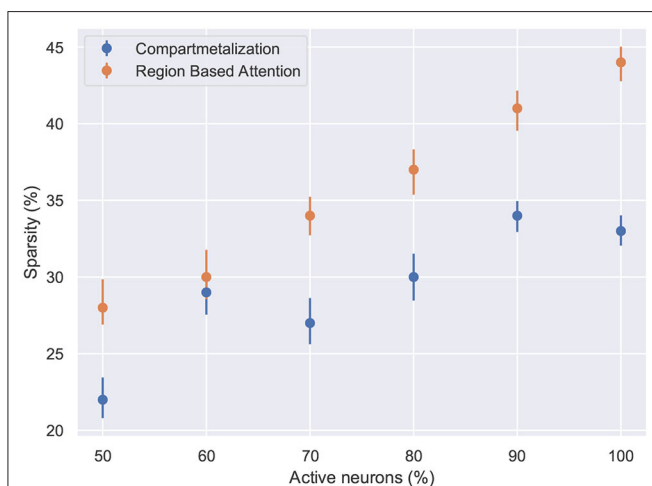


FIGURE 7 | The network sparsity observed after one epoch with respect to the number of active neurons in the network. Compartmentalization is able to find an optimal space despite the variance in the network inactivity and it is observed that the optimal sparsity is achieved at $\approx 30\%$ for compartmentalized ModNet and $\approx 40\%$ for ModNet with attention (Evaluated on the MNIST dataset).

learning using multiple learning rules. However, it compensates by showing improved performance. Enforcing sparsity in the network enables better distribution of information and thereby compartmentalization can be extended to a multi task learning scenario with different subsets of compartments responding to different tasks.

4.2. Few Shot Learning Tasks

One criticism of ModNet and the proposed learning mechanisms is regarding their ability to perform well on larger and more complex problems. Therefore, to evaluate the advantages of using

heterosynaptic interactions for complex tasks, we select the few shot learning task.

4.2.1. Benchmarks and Experimental Setup

The modulatory trace learning rule is tested for few shot learning on the Omniglot corpora. The Omniglot dataset Lake et al. (2015), as shown in **Figure 8** contains examples from 50 alphabets ranging from well-established international languages to lesser known local dialects. It consists of a total of 1,623 characters with letters from each alphabet varying from about 15 to upwards of 40 characters. Each of these are hand drawn by 20 different people. Moreover, each character in Omniglot is a 105×105 binary image. Thus the dataset has 1,623 classes with 20 (105×105 images) examples per class.

Figure 9 shows the baseline CNN architecture used for few shot learning task. It consists of a stack of modules, each of which is a 3×3 convolution with 64 filters followed by batch normalization, a ReLU non-linear activation function, and a 2×2 max-pooling layer. The images are resized to 28×28 so that, when 4 modules are stacked, the resulting feature map is $1 \times 1 \times 64$. This output is a 64 sized vector which then feeds into a N-way softmax layer. Concurrently, the label of the target character is fed as a one-hot encoded value to the softmax layer, guiding the correct output when a label is present.

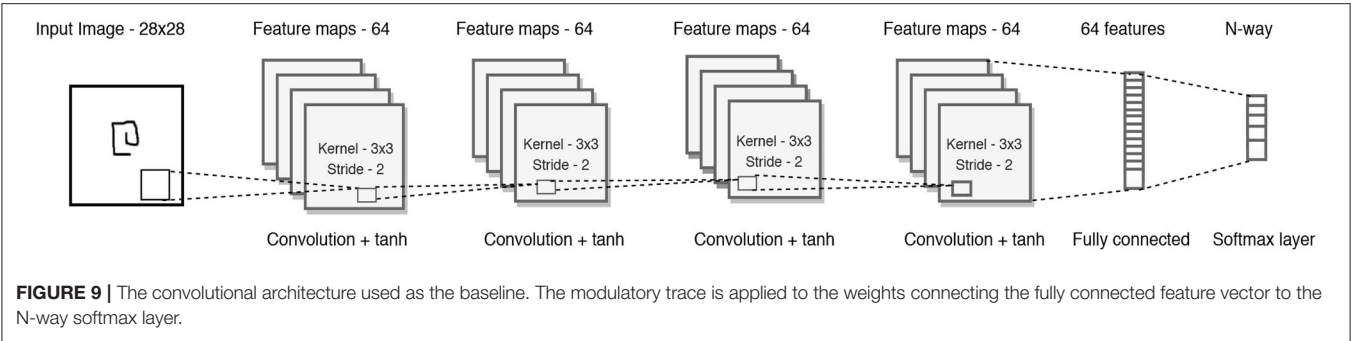
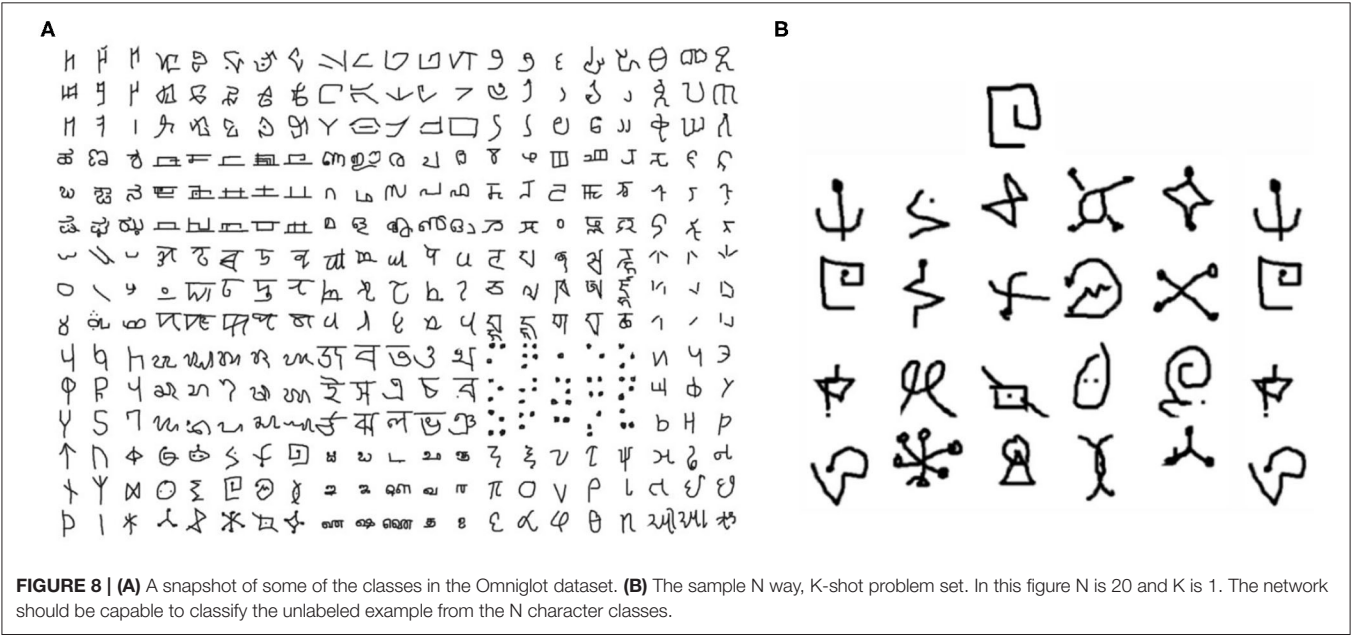
The task is modeled as an N-way, k-shot classification setup, similar to most of the previous works for few shot classification. This problem thus can be formalized as follows: pick N unseen character classes and K examples of each class from those N, independent of the alphabet and let that set be N, K (Vinyals et al., 2016) as shown in **Figure 8B**. Each of these instances together with the class labels (from 1 to N) are presented to the model. Then a new unlabeled instance from one of the N classes is presented to the model. The model's performance is defined as the model's accuracy in classifying this unlabeled example. The baseline configuration is tested for N = 5 and K = 1 (five way, one shot learning).

The effect of modulatory mechanisms in deeper networks is evaluated for the task of few shot learning on the omniglot dataset (Lake et al., 2015). Modulatory plasticity is introduced in the baseline architecture in **Figure 9** for the weights connecting the final feature layer to the softmax layer. The rest of the convolutional embedding does not have modulatory plasticity associated to it. Thus, across the training episodes, the convolutional architecture is expected to learn an adequate discriminant between arbitrary handwritten characters. Meanwhile, the weights between the convolutional network and the softmax should learn to memorize associations between observed patterns and outputs, which are directly influenced by the modulatory context and the labels. **Table 3** lists details on the network configuration and hyperparameters used for few-shot learning task. Similar to previous works (Finn et al., 2017; Mishra et al., 2017; Miconi et al., 2018), the dataset is divided into 1,523 classes for training and 100 classes for testing. The network is trained using an Adam optimizer with a learning rate of 3×10^{-5} , multiplied by 2/3 every 100,000 episodes) over 500,000 episodes. To evaluate the final performance, multiple models with different

TABLE 2 | Ablation study of the proposed learning methods for ModNet on the MNIST and the Fashion-MNIST datasets.

Network	Datasets					
	MNIST			F-MNIST		
	Samples ^a	Train Acc ^b	Test Acc ^c	Samples	Train Acc	Test Acc
ModNet	8,000	92.87	87.25	12,000	80.87	74.13
ModNet + Attention	9,500	91.46	86.47	13,000	78.65	72.93
ModNet + Attention + Compartment	11,000	93.82	90.86	15,500	90.85	84.73
ModNet + Sparsity*	14,000	91.39	85.82	23,000	78.76	74.27

*The network is initialized with 30% sparse input and output connections.
^aRefers to the number of samples to reach initial convergence.
^bThe mean training accuracy observed where the network initially converges.
^cThe test accuracy of the network when trained for the number of samples specified.



random seed initializations are trained and then tested on the previously unseen classes for 200 episodes.

4.2.2. Performance Analysis

To understand how few shot learning networks train and how the loss varies when presented with new distributions of inputs in a N-way,k-shot learning task, the moving average of median

loss, and the mean loss across the training procedure is visualized. **Figure 10A** shows the moving average of the median loss across multiple runs. The mean is calculated across points after different milestones spaced evenly for every 50,000 episodes. The average loss is computed and saved after 100 episodes to create a net loss matrix. The median of losses along the milestone axis is computed and the moving average along those median values

is plotted. The overall median loss decreases as the network is trained longer. This plot shows how few shot learning task does not have a standard loss gradient when presented with more samples as the training involves accessing new set of inputs every time.

Figure 10B represents how the loss actually varies during the training phase. The repeated spikes in the loss is due to the way the network is trained. Since, there are 1,523 classes and 5 classes are randomly selected and moreover, randomly permuted and rotated, the high loss scenarios occur mostly when the randomly selected samples are almost similar. Thus, the loss is higher in some episodes as a result of being trained on new and difficult episodes. The 5 samples have very few distinctive features and thus learning the associations might be difficult in that case. This type of learning is consistent across multiple runs with different random initializations as shown in the plot.

The results in **Figure 11** show the performance of the proposed network in comparison to the accuracies reported in the recent literature. Other than the memory networks, all the other works make use of the baseline convolutional network that has been described previously. The accuracy of the model is almost similar to the computationally intensive MAML (Finn et al., 2017) approach which optimizes for the loss function

TABLE 3 | Experimental setup and hyperparameters for evaluating the modulatory trace learning rule.

Network features		Modulatory trace learning	
Datasets		Omniglot	
Network size		4 Conv, 1 FC	
Hyperparameters	LR	γ	δ
	3×10^{-5}	0.02	0.01
Training episodes		500,000	
Runs		5	
Data augmentation		Rotations (Multiples of 90°)	

using gradient descent. The results are almost similar to the Matching networks (Vinyals et al., 2016), Differentiable Plastic networks (Miconi et al., 2018) and Meta networks (Munkhdalai and Yu, 2017). The results reported in SNAIL (Mishra et al., 2017) outperform all the other networks and are currently state of the art but the difference is barely significant. The SNAIL approach trains a whole temporal convolutional layer and causal layers on top of the baseline convolutional embedding leading to a significant increase in the number of parameters. The proposed network performance is near state of the art accuracy for an additional 330 ($66 \times N$, with $N=5$) parameters. The network has a total of 112,065 parameters. The proposed model requires $\approx 10x$ fewer training episodes than Differentiable Plastic networks.

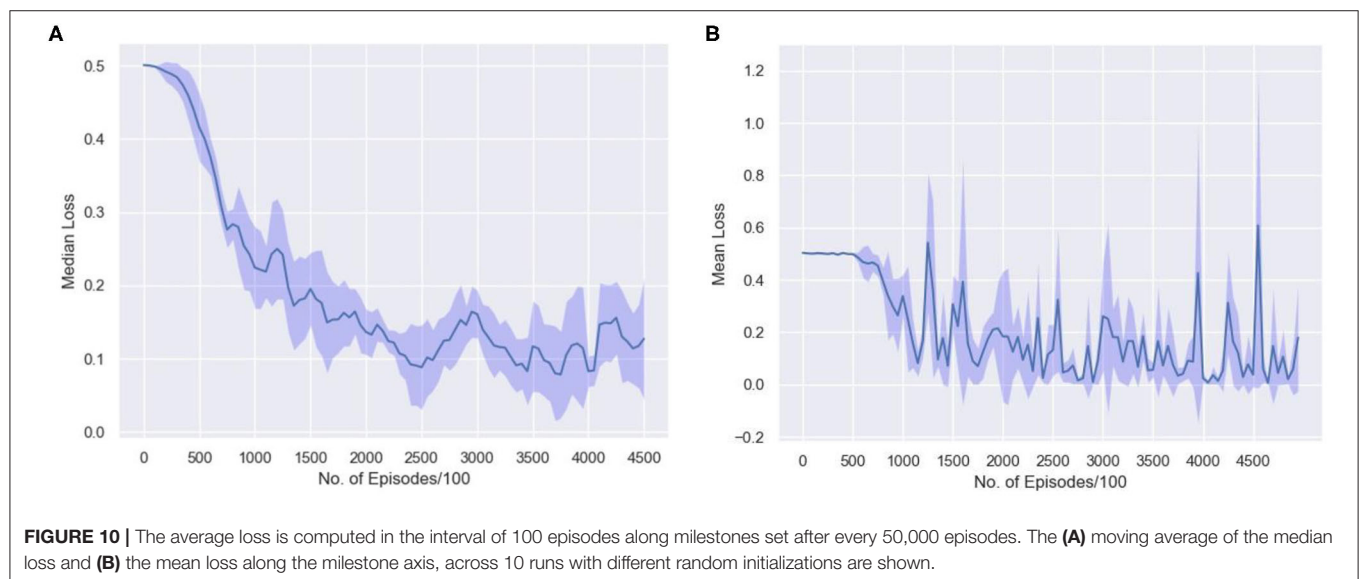
4.2.3. Resource Efficiency Through Quantization

To further optimize the resource usage in ModNet, we study the impact of reducing the bit precision. The convolutional embedding is pretrained offline and few-shot learning is performed on a quantized version of the model.

$$PLA(x) = \begin{cases} 1, & \text{when } abs(x) \geq 3 \\ 0.06abs(x) + 0.815, & \text{when } 1.5 \leq x < 3 \\ 0.443abs(x) + 0.24, & \text{when } 0.5 \leq abs(x) < 1.5 \\ 0.924abs(x), & \text{otherwise} \end{cases} \quad (15)$$

$$Tanh(x) = \begin{cases} PLA(x), & \text{when } x \geq 0 \\ -PLA(x), & \text{otherwise} \end{cases} \quad (16)$$

The features in the quantized model include inputs and weights represented in 16-bit fixed-point format with 6 bits for signed integer and 10 bits for fractional part, along with 32-bit partial sum accumulators which are again rounded to 16-bit. The tanh activation is replaced by a piecewise linear approximation as shown in Equations (15) and (16). One interesting observation is that the bit-precision of the output layer made a significant impact on the accuracy. The network performance degraded



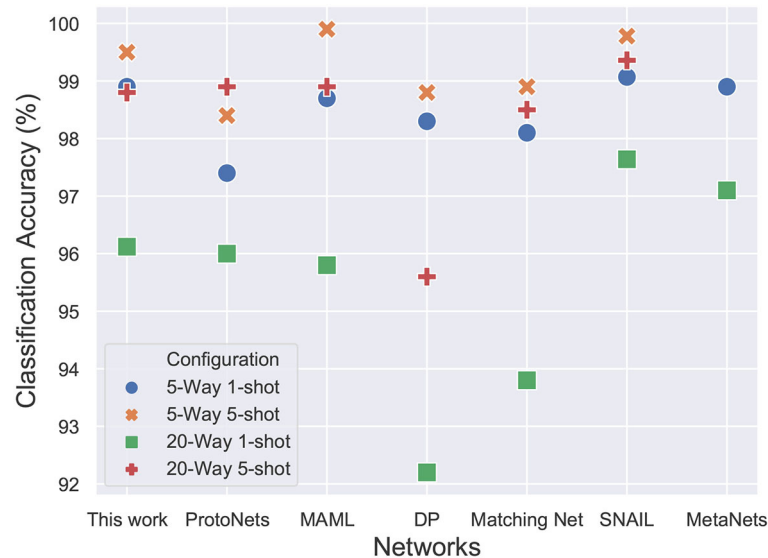


FIGURE 11 | 20-way and 5-way, 1-shot and 5-shot classification accuracies on Omniglot dataset. The proposed model achieves comparable accuracy to state-of-the-art SNAIL architecture while requiring 20x fewer trainable parameters and fewer training episodes as compared to MAML.

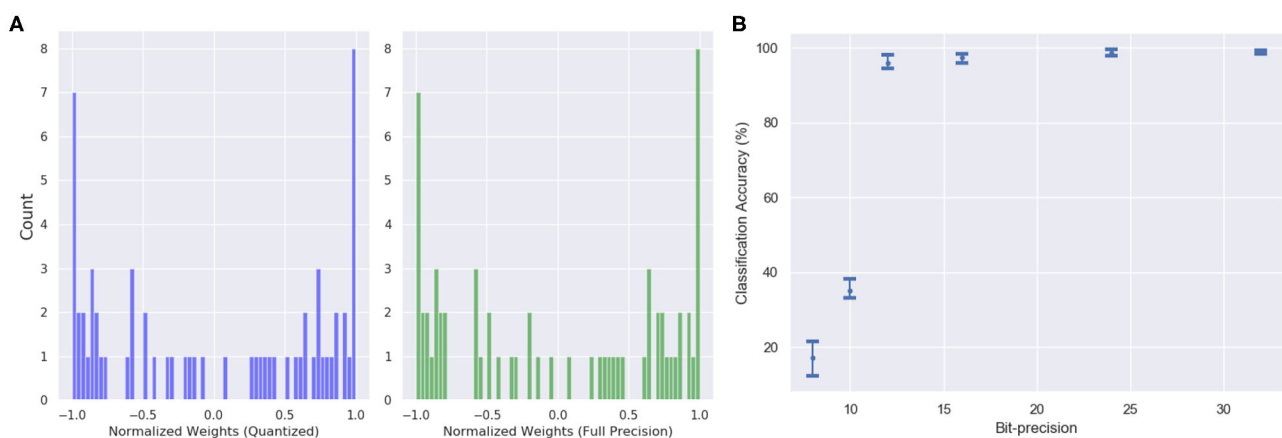


FIGURE 12 | (A) The weight distribution of the output feature vector for 16-bit fixed point quantized weights (blue) and 32-bit floating point precision weights (green). **(B)** Variation in network performance with respect to changing bit precision. The performance degradation is minimal for 12-bit precision weights and inputs.

substantially (from ≈ 98 to 47%) when quantized to 16-bit fixed point. However, the accuracy remained the same as the 32-bit floating point when the output layer is not quantized. **Figure 12** shows the weight distribution of the output feature vector for the quantized and the 32-bit operations. The plot shows that the quantized version is able to generate almost similar output feature activations with low quantization error of 1.254%.

5. CONCLUSIONS

This work shows that incorporating neuromodulatory mechanisms in neural networks is effective toward realizing dynamic learning systems that are able to learn associations and discrimination in the input stream, based on a context. The main contributions of this work are the design of an architecture and

adaptive learning rules to introduce modulatory dynamics in the neural networks. The proposed architecture uses simple plasticity rules with a modulatory control mechanism for learning instead of using backpropagation. The ModNet architecture is capable of learning quickly and from fewer samples. This work also introduces a neuromodulation-inspired training technique to self-modify weights in a network. These simple plasticity mechanisms when combined with conventional gradient descent approaches are able to solve non-trivial tasks like few shot learning of different human written characters.

ModNet and compartmentalized ModNet, despite being shallow networks are able to train on the complete MNIST and Fashion-MNIST dataset in just 2 epochs and reach convergence within 8,000 samples of MNIST with 91% accuracy and 12,000 samples of Fashion-MNIST with 89%

accuracy. Furthermore, the modulatory trace learning rule in tandem with backpropagation shows accuracy of 98.8% with a 95% confidence interval on the non-trivial few shot learning task on the Omniglot dataset for an additional 325 trainable parameters. These experiments prove that compact and simple meta learning approaches via neuromodulation can perform as well as current computationally intensive methods. Compartmentalization integrated with multiple local plasticity rules might alleviate catastrophic forgetting in neural networks and enable multi-task learning.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: <http://yann.lecun.com/exdb/mnist/>, <https://github.com/zalandoresearch/fashion-mnist>, <https://github.com/brendenlake/omniglot>.

REFERENCES

- Aso, Y., Sitaraman, D., Ichinose, T., Kaun, K. R., Vogt, K., Belliard-Guérin, G., et al. (2014). Mushroom body output neurons encode valence and guide memory-based action selection in drosophila. *eLife* 3:e04580. doi: 10.7554/eLife.04580
- Bargmann, C. I. (2012). Beyond the connectome: how neuromodulators shape neural circuits. *Bioessays* 34, 458–465. doi: 10.1002/bies.201100185
- Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*.
- Carew, T. J., Walters, E. T., and Kandel, E. R. (1981). Classical conditioning in a simple withdrawal reflex in *Aplysia californica*. *J. Neurosci.* 1, 1426–1437. doi: 10.1523/JNEUROSCI.01-12-01426.1981
- Changizi, M. A. (2013). *The Brain from 25,000 Feet: High Level Explorations of Brain Complexity, Perception, Induction and Vagueness* (Springer Science & Business Media). doi: 10.1007/978-94-017-0293-5
- Chittka, L., and Niven, J. (2009). Are bigger brains better? *Curr. Biol.* 19, R995–R1008. doi: 10.1016/j.cub.2009.08.023
- Cooper, S. J. (2005). Donald O. Hebb's synapse and learning rule: a history and commentary. *Neurosci. Biobehav. Rev.* 28, 851–874. doi: 10.1016/j.neubiorev.2004.09.009
- Daram, A. R., Kudithipudi, D., and Yanguas-Gil, A. (2019). "Task-based neuromodulation architecture for lifelong learning," in *20th International Symposium on Quality Electronic Design (ISQED)* (IEEE), 191–197. doi: 10.1109/ISQED.2019.8697362
- Decker, M. W., and McGaugh, J. L. (1991). The role of interactions between the cholinergic system and other neuromodulatory systems in learning and memory. *Synapse* 7, 151–168. doi: 10.1002/syn.890070209
- Doya, K. (2002). Metalearning and neuromodulation. *Neural Netw.* 15, 495–506. doi: 10.1016/S0893-6080(02)00044-8
- 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* (Sydney, NSW: JMLR), 1126–1135. doi: 10.5555/3305381.3305498
- Frémaux, N., and Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Front. Neural Circ.* 9:85. doi: 10.3389/fncir.2015.00085
- Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *J. Mach. Learn. Res.* 3, 115–143. doi: 10.1162/153244303768966139
- Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: a search space odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 2222–2232. doi: 10.1109/TNNLS.2016.2582924
- Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron* 95, 245–258. doi: 10.1016/j.neuron.2017.06.011
- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). "Densely connected convolutional networks," in *CVPR* (Honolulu, HI), 3. doi: 10.1109/CVPR.2017.243
- Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2004). Extreme learning machine: a new learning scheme of feedforward neural networks. *International Joint Conference on Neural Networks* 2, 985–990. doi: 10.1109/ijcnn.2004.1380068
- Kar, A. K. (2016). Bio inspired computing-a review of algorithms and scope of applications. *Expert Syst. Appl.* 59, 20–32. doi: 10.1016/j.eswa.2016.04.018
- Katz, P., and Edwards, D. (1999). "Metamodulation: the control and modulation of neuromodulation," in *Beyond Neurotransmission: Neuromodulation and Its Importance for Information Processing*, ed P. S. Katz, 349–381. doi: 10.1093/acprof:oso/9780198524243.003.0010
- Katz, P. S. (1999). *Beyond Neurotransmission: Neuromodulation and Its Importance for Information Processing*. Oxford, UK: Oxford University Press.
- Koch, G., Zemel, R., and Salakhutdinov, R. (2015). "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning Workshop* (Lille Grande Palais: Lille).
- Kolouri, S., Ketz, N., Zou, X., Krichmar, J., and Pilly, P. (2019). Attention-based selective plasticity. *arXiv preprint arXiv:1903.06070*.
- Kondo, T. (2007). Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control. *Appl. Soft Comput.* 7, 189–202. doi: 10.1016/j.asoc.2005.05.004
- Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science* 350, 1332–1338. doi: 10.1126/science.aab3050
- Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: a convolutional neural-network approach. *IEEE Trans. Neural Netw.* 8, 98–113. doi: 10.1109/72.554195
- LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 2278–2324. doi: 10.1109/5.726791
- Lihoreau, M., Latty, T., and Chittka, L. (2012). An exploration of the social brain hypothesis in insects. *Front. Physiol.* 3:442. doi: 10.3389/fphys.2012.00442
- Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356
- Miconi, T., Clune, J., and Stanley, K. O. (2018). Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv preprint arXiv:1804.02464*.
- Miconi, T., Rawal, A., Clune, J., and Stanley, K. O. (2020). Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. *arXiv preprint arXiv:2002.10585*.
- Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2017). A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*.

AUTHOR CONTRIBUTIONS

AD developed the theory and performed the experiments and computations. DK and AY-G verified the experiments and supervised the findings in the proposed work. All authors contributed to the final manuscript.

FUNDING

This work was partially supported through the Lifelong Learning Machines (L2M) program from DARPA/MTO.

- Montgomer, S. H., Currie, A., Lukas, D., Boogert, N., Buskell, A., Cross, F. R., et al. (2018). Ingredients for understanding brain and behavioral evolution: ecology, phylogeny, and mechanism. *Comparative Cognition & Behavior Reviews* 13, 99–104. doi: 10.3819/CCBR.2018.130011
- Munkhdalai, T., and Yu, H. (2017). “Meta networks,” in *Proceedings of Machine Learning Research* (Sydney, NSW: International Convention Centre), 2554–2563. Available online at: <http://proceedings.mlr.press/v70/munkhdalai17a.html>
- Pedrosa, V., and Clopath, C. (2017). The role of neuromodulators in cortical plasticity: a computational perspective. *Front. Synapt. Neurosci.* 8:38. doi: 10.3389/fnsyn.2016.00038
- Roberts, A. C., and Glanzman, D. L. (2003). Learning in aplysia: looking at synaptic plasticity from both sides. *Trends Neurosci.* 26, 662–670. doi: 10.1016/j.tins.2003.09.014
- Rosenfeld, A., Zemel, R., and Tsotsos, J. K. (2018). The elephant in the room. *arXiv preprint arXiv:1808.03305*.
- Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). “Backpropagation: the basic theory,” in *Backpropagation: Theory, Architectures and Applications* (L. Erlbaum Associates Inc.), 1–34. doi: 10.5555/201784.201785
- Sak, H., and Senior, A. W. (2018). Processing acoustic sequences using long short-term memory (lstm) neural networks that include recurrent projection layers. *US Patent App. 10/026397*. Google Patents.
- Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*.
- Shine, J. M., Breakspear, M., Bell, P. T., Martens, K. A. E., Shine, R., Koyejo, O., et al. (2019). Human cognition involves the dynamic integration of neural activity and neuromodulatory systems. *Nat. Neurosci.* 22:289. doi: 10.1038/s41593-018-0312-0
- 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
- Soltoggio, A., Bullinaria, J. A., Mattiussi, C., Dürr, P., and Floreano, D. (2008). “Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios,” in *Proceedings of the 11th International Conference on Artificial Life (Alife XI)* (MIT Press), 569–576.
- Sutton, R. S., and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. Vol. 135. Cambridge: MIT Press.
- Theobald, J. (2014). Insect neurobiology: how small brains perform complex tasks. *Curr. Biol.* 24, R528–R529. doi: 10.1016/j.cub.2014.04.015
- Vinyals, O., Blundell, C., Lillicrap, T., and Wierstra, D. (2016). “Matching networks for one shot learning,” in *Advances in Neural Information Processing Systems* (Red Hook, NY: Curran Associates Inc.), 3637–3645. doi: 10.5555/3157382.3157504
- Williams, R. J., and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* 1, 270–280. doi: 10.1162/neco.1989.1.2.270
- Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.
- Yanguas-Gil, A., Mane, A., Elam, J. W., Wang, F., Severa, W., Daram, A., et al. (2019). “The insect brain as a model system for low power electronics and edge processing applications,” in *2019 IEEE Space Computing Conference (SCC)* (Pasadena, CA: IEEE), 60–66. doi: 10.1109/SpaceComp.2019.00012

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.

Copyright © 2020 Daram, Yanguas-Gil and Kudithipudi. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



Bitstream-Based Neural Network for Scalable, Efficient, and Accurate Deep Learning Hardware

Hyeonuk Sim^{1,2} and Jongeun Lee^{1,2*}

¹ School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, South Korea,

² Neural Processing Research Center, Seoul National University, Seoul, South Korea

OPEN ACCESS

Edited by:

Kaushik Roy,
Purdue University, United States

Reviewed by:

Yam Song Chua,
Huawei Technologies, China
Aayush Ankit,
Purdue University, United States

*Correspondence:

Jongeun Lee
jlee@unist.ac.kr

Specialty section:

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

Received: 17 March 2020

Accepted: 27 October 2020

Published: 23 December 2020

Citation:

Sim H and Lee J (2020)
Bitstream-Based Neural Network for
Scalable, Efficient, and Accurate Deep
Learning Hardware.
Front. Neurosci. 14:543472.
doi: 10.3389/fnins.2020.543472

While convolutional neural networks (CNNs) continue to renew state-of-the-art performance across many fields of machine learning, their hardware implementations tend to be very costly and inflexible. Neuromorphic hardware, on the other hand, targets higher efficiency but their inference accuracy lags far behind that of CNNs. To bridge the gap between deep learning and neuromorphic computing, we present bitstream-based neural network, which is both efficient and accurate as well as being flexible in terms of arithmetic precision and hardware size. Our bitstream-based neural network (called SC-CNN) is built on top of CNN but inspired by stochastic computing (SC), which uses bitstreams to represent numbers. Being based on CNN, our SC-CNN can be trained with backpropagation, ensuring very high inference accuracy. At the same time our SC-CNN is deterministic, hence repeatable, and is highly accurate and scalable even to large networks. Our experimental results demonstrate that our SC-CNN is highly accurate up to ImageNet-targeting CNNs, and improves efficiency over conventional digital designs ranging through 50–100% in operations-per-area depending on the CNN and the application scenario, while losing <1% in recognition accuracy. In addition, our SC-CNN implementations can be much more fault-tolerant than conventional digital implementations.

Keywords: bitstream-based neural network, neuromorphic computing, stochastic computing, deep learning hardware, dynamic precision scaling, SC-CNN, variable precision

1. INTRODUCTION

In a broad sense of the term, *neuromorphic system* refers to a system engineered based on the organizing principles of the nervous system (Mead, 1990). For instance, a CMOS transistor's I-V curve follows an exponential curve under specific conditions and the amount of charge in a capacitor is the time integration of current. Thus, if a system's computation mostly consists of the elementary operations directly derived from the physical principles of devices, such as exponential and time integration, extremely efficient systems can be built by using those elementary operations of devices, as opposed to using AND and OR primitives, which is an artifact of digital design principle (Mead, 1990). Along the same line, neuromorphic system also means mimicking the *structure*, in addition to the behavior, of the nervous system, which is argued to be an important ingredient to attaining desirable system properties, such as high energy efficiency and error resilience, which may be as essential as accuracy in biological nervous systems (Hawkins and George, 2006).

On the other hand, neural networks in the neuromorphic camp, exemplified by the spiking neural networks (SNNs) (Lee et al., 2016), are criticized for their low performance; their inference accuracy lags far behind that of artificial neural networks used in deep learning (Roy et al., 2019). This criticism is a serious one, since the deep learning's efficiency, which is generally regarded as low compared with that of SNNs and other neuromorphic-based ones, can be improved a lot, if inference accuracy can be sacrificed. For instance, low precision networks (Zhou et al., 2016) and binary/ternary networks (Courbariaux and Bengio, 2016; Hubara et al., 2017) have far less computation compared with the original models at the cost of slight accuracy loss. One may argue that the apparent performance advantage of deep learning comes partly from the dataset itself, given the pivotal role played by datasets in the evolution of deep learning (Roy et al., 2019). Nevertheless, it is clear that there is a widening gap between neuromorphic computing and deep learning in terms of accuracy, which must be remedied to make the neuromorphic approach more appealing.

To bridge the gap between neuromorphic computing and deep learning, we present a class of neural networks based on bitstreams. Our neuron model is inspired by stochastic computing (SC), which is an alternative design principle for hardware that is distinguished from both digital and analog design principles. In SC, a number is represented as a bitstream that can be carried on a single wire over a period of time, which is reminiscent of analog computing, yet at the same time, SC circuits can be entirely made out of digital components. The advantages of SC over digital implementations, such as low-cost computation (e.g., multiplication), flexible precision, and high error resilience has led to many applications in image processing and neural networks (Kim et al., 2016; Li et al., 2016, 2017a,b; Ren et al., 2016, 2017; Sim et al., 2017; Zhakataev et al., 2018). However, the conventional SC as applied to neural networks suffers from the low accuracy problem especially with large neural networks, much like neuromorphic computing.

Our bitstream-based neural network, which is based on convolutional neural network (CNN) and hence called SC-CNN, addresses the low accuracy problem of SC while retaining its main advantages. At the same time, our neural network is deterministic, hence repeatable, and is highly accurate and scalable even to large networks¹. Much like any CNN, our SC-CNN can be trained with backpropagation, ensuring very high inference accuracy. In addition, as CNNs grow more complex and diverse, there is a need for a more reconfigurable hardware architecture that can run various CNNs with different precision requirements at high efficiency. Such reconfigurable precision, which we call *dynamic precision scaling* (DPS), is particularly useful for SC, where 1-bit saving can reduce computation latency by 50% (see **Figure 1**), suggesting a great potential for higher efficiency on CNNs with diverse precision requirements.

In this paper, we present SC-CNN, which is highly optimized for both accuracy and efficiency as well as flexibility, such as

dynamically adjustable precision. Specifically, we first propose dynamic precision scaling for SC-CNN, which extends our previous work (Sim and Lee, 2017) such that the precision of input/output data can be arbitrarily modulated at runtime (see section 3). The extension has very little overhead and allows us to be efficiently parsimonious with regard to precision, which gives a considerable reward in latency saving. Second, in terms of allocating precision across the value range of a variable, we observe that in some layers input activations are always non-negative, meaning that we can reduce precision by 1 bit (with corresponding 50% latency saving) and still get effectively the same accuracy. We call this optimization *half-range specialization* (HRS). Importantly, we implement the HRS optimization as an add-on feature that can be switched on dynamically, so that the same hardware can run all layers of a CNN regardless of the input range (see section 3.4). Third, we present our design methodology to optimize precision across layers of a CNN in section 4. Fourth, exploiting the compactness of our neuron we explore multi-dimensional parallelism to better utilize a given area budget. This is motivated by the fact that the nervous system does no time-multiplexing but implements all neurons with dedicated resources, which is likely an important factor of its high efficiency. We show that multi-dimensional parallelism can give a significant efficiency improvement for SC-CNN over using limited parallelism.

Our experimental results demonstrate that our SC-CNN can be as efficient as conventional digital designs up to ImageNet-targeting CNNs, such as AlexNet (Krizhevsky et al., 2012) and GoogleNet (Szegedy et al., 2015), with <1% degradation in recognition accuracy. This is quite significant as the previous work on SC-CNN was only able to show it up to Cifar-10 (Sim and Lee, 2017), which is much smaller than ImageNet. Second, we show that our SC-CNN can be over 100% more efficient in terms of operations-per-area over conventional digital design when the same hardware is used for multiple CNN applications of varying precision requirements. Third, we show that even for a single application scenario where the hardware is designed only for one application (e.g., AlexNet), our SC-CNN can still be 52% more efficient than conventional digital design. Fourth, we show that our neuromorphic SC-CNN is very scalable, achieving very high efficiency at a high throughput level; more specifically, our 4D-parallel neuromorphic SC-CNN can give nearly 100 times better efficiency in ADP (area-delay product) over 2D-parallel architectures. Fifth, our error injection experiments demonstrate that our SC-CNNs can be significantly more fault-tolerant than conventional digital implementations. These results suggest that our SC-CNN, which has several traits of neuromorphic computing, such as compact and efficient neurons, flexible precision, and high fault tolerance, can still be highly accurate and scalable similar to deep learning models.

The rest of the paper is organized as follows. After reviewing the related work in section 2, we present the SC-CNN and neuron-level optimizations in section 3. In section 4, we present the network-level optimizations including those targeting neuromorphic applications. In section 5, we present our experimental results, and section 6 concludes the paper with a summary of the work and future directions.

¹Despite the deterministic nature of our SC-MAC, we retain the term, SC-MAC, in this paper because its operation is based on bitstreams. It also helps maintain consistency with previous work.

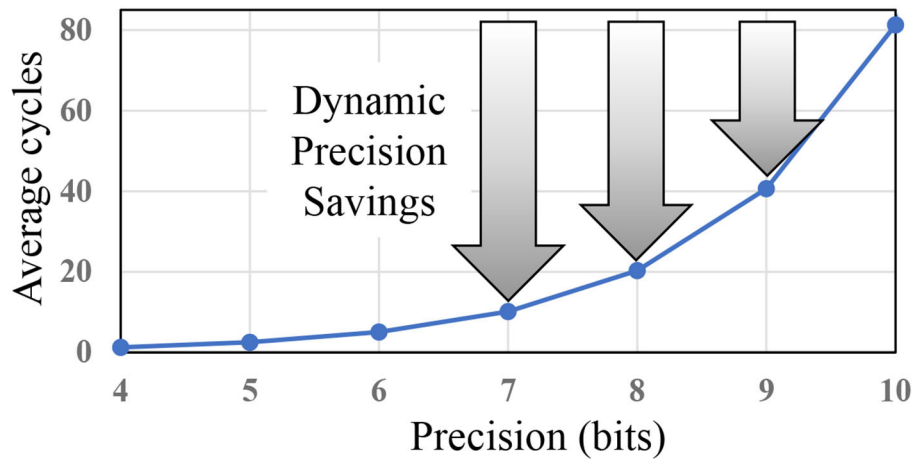


FIGURE 1 | Dynamic precision, or using lower precision whenever possible, can give huge boost in efficiency for stochastic computing (SC).

2. RELATED WORK

2.1. SC-Based Neural Network

The most closely related to our work is SC-based neural networks. Thanks to low implementation cost and high error resilience, SC is seen as a promising approach to accelerating applications in certain domains including image processing and neural network. Previous work on SC-based acceleration of CNNs can be classified into two categories: fully parallel and tile-based.

In the fully parallel approach (Kim et al., 2016; Li et al., 2016, 2017a,b; Ren et al., 2016, 2017), all neurons are implemented spatially using dedicated hardware resources, and they operate in parallel such that the neural network circuit will produce output as the wave of input data sweeps through the circuit. It could have very high energy efficiency owing to the fact that it does not involve external memory access to store intermediate result, but has limited applicability because it cannot support arbitrarily large CNNs.

The tile-based approach (Sim and Lee, 2017; Sim et al., 2017), on the other hand, is more scalable in terms of the number of layers and neurons supported, since it works by tiling the computation of a layer into smaller fixed-sized arrays, each of which is performed by the same hardware block. This is also the approach employed by all recent hardware accelerators for CNNs (Chen et al., 2014, 2017). The intermediate results are saved to and reloaded from buffers, which are typically on-chip and backed by external memories. Now for efficiency reasons in terms of storage and external memory bandwidth, SC data should be saved in memories as conventional digital numbers, in which case every on-chip memory access would require a conversion between SC and digital representations, which is a considerable overhead in the tile-based approach.

A recent technique (Sim and Lee, 2017) proposes a new SC-MAC (multiply and accumulate) algorithm by combining the SNG (stochastic number generator), SC multiply operation, and an addition in the digital domain. This input and output of the

SC-MAC is conventional digital. As such, it fits nicely with the tile-based approach, greatly reducing the conversion overhead. However, the inherent precision disadvantage of SC—that SC requires exponentially longer bitstreams as precision increases—has so far kept SC from being competitive on larger CNNs, such as AlexNet (Krizhevsky et al., 2012).

In terms of accuracy, both the fully parallel (Kim et al., 2016; Yu et al., 2017) and tile-based approaches (Sim and Lee, 2017; Sim et al., 2017) have shown competitive result against conventional digital designs for small CNNs with about 10 classification categories, such as MNIST (LeCun et al., 2010) and CIFAR-10 (Krizhevsky and Hinton, 2009). It seems that the retraining capability of CNNs helps cope with the approximating nature of stochastic computing.

2.2. Neuromorphic Computing

Neuromorphic computing is multi-faceted. On the one hand, there is the neuromorphic engineering approach (Mead, 1990), where researchers try to design useful systems based on the elementary operations of devices, which could lead to much more efficient systems than conventional digital designs. On the other hand, many neuromorphic models (Izhikevich, 2003) and systems (Markram, 2012) aim to imitate or emulate the nervous system as faithfully as possible, which can aid, e.g., with brain scientists studying the nervous system. Some of the effort has led to the design of dedicated hardware chips, such as TrueNorth (Akopyan et al., 2015) and Loihi (Davies et al., 2018).

Due to the similarity between neuromorphic computing and deep learning, there is a hope that the neuromorphic approach can 1 day lead to a much better neuron model than what is used today. For instance, spiking neuron, such as leaky integrate-and-fire (LIF) model (Maass, 1997) is referred to as the third-generation neuron model after McCulloch-Pitts neuron (McCulloch and Pitts, 1943) and perceptron (Hornik et al., 1989). Since spiking neurons generally use *timing* information, they may be able to achieve sparse, event-driven neural networks

with much higher energy efficiency than perceptron-based neural networks used in today's deep learning (Roy et al., 2019). However, it remains to be seen whether SNNs can indeed show competitive performance as deep learning models on large datasets. To improve the efficiency of SNNs, a recent technique (Stöckl and Maass, 2020) reduces the number of spikes a neuron has to emit to pass information to subsequent neurons, by essentially encoding the information as a binary number where each bit position has a different weight. However, it discusses no hardware implementation, thus with no area- or energy-efficiency result. To improve the accuracy of SNNs, which is another way to improve efficiency, a training method (Wu et al., 2019) was proposed that uses an auxiliary artificial neural network that approximates the behavior of the coupled SNN and thereby enables back-propagation-based training for the SNN. Our training method is also based on back-propagation, which is why our models can show very high accuracy. Note that the training method, such as Wu et al. (2019) does not eliminate the need for (re)training for SNNs, but it only makes SNN training converge better and faster.

2.3. Deep Learning Hardware

Recent CNN hardware implementations (Chen et al., 2014, 2017; Han et al., 2016) all have their precision fixed at design time. This has an obvious disadvantage when the application's precision requirement is different from the designed precision, that is, accuracy loss (when the needed precision is higher) and efficiency loss (when it is lower). This mismatch can happen even within a single CNN, such as varying precision requirement among different layers (Judd et al., 2016), which causes some inefficiency even when the accelerator is running the CNN for which it is designed. One solution to the precision mismatch problem is to use bit-serial hardware, such as bit-serial multiplier (Judd et al., 2016). Our solution can provide an alternative solution, which has other advantages, such as error resiliency (see section 5.7) as well as being more efficient than the bit-serial approach (see section 5.4).

In comparison, SC can be more efficient partially owing to the optimizations proposed in this paper. This helps close the efficiency gap between SC and bit-parallel conventional digital for a wider range of precisions as we show later in **Figure 7**, while still retaining the benefits of SC, such as DPS. Also the effect of DPS could be higher in SC than in conventional digital, since 1-bit reduction in SC may reduce the delay of computation by about 50% as shown in **Figure 1**. In the figure, the y -axis shows computation delay, which is given by the bitstream length needed to deliver the precision on the x -axis. The exponential relationship holds for both the conventional SC (Kim et al., 2016) and the new SC-MAC (Sim and Lee, 2017).

3. DYNAMIC PRECISION SCALING SC-CNN

Our neuron model is built on top of SC-MAC (Sim and Lee, 2017), which uses a deterministic and optimized SC multiply algorithm. We first briefly review it and its application to neural

networks, and present two extensions: (i) DPS optimization that allows the hardware precision to vary dynamically with little overhead, and (ii) HRS optimization that can be dynamically enabled.

3.1. Analysis of Baseline SC-MAC

The key features of the baseline SC-MAC (Sim and Lee, 2017) include SNG integration, a novel SC multiply algorithm, and variable latency, which help it achieve superior efficiency as compared with conventional SC-based MACs, such as Kim et al. (2016). The MAC unit takes two operands labeled x and w , and generates output y that should approximate xw . All the inputs/output are represented as conventional digital, as illustrated in **Figure 2** (shown in red is for dynamic precision discussed in the next section).

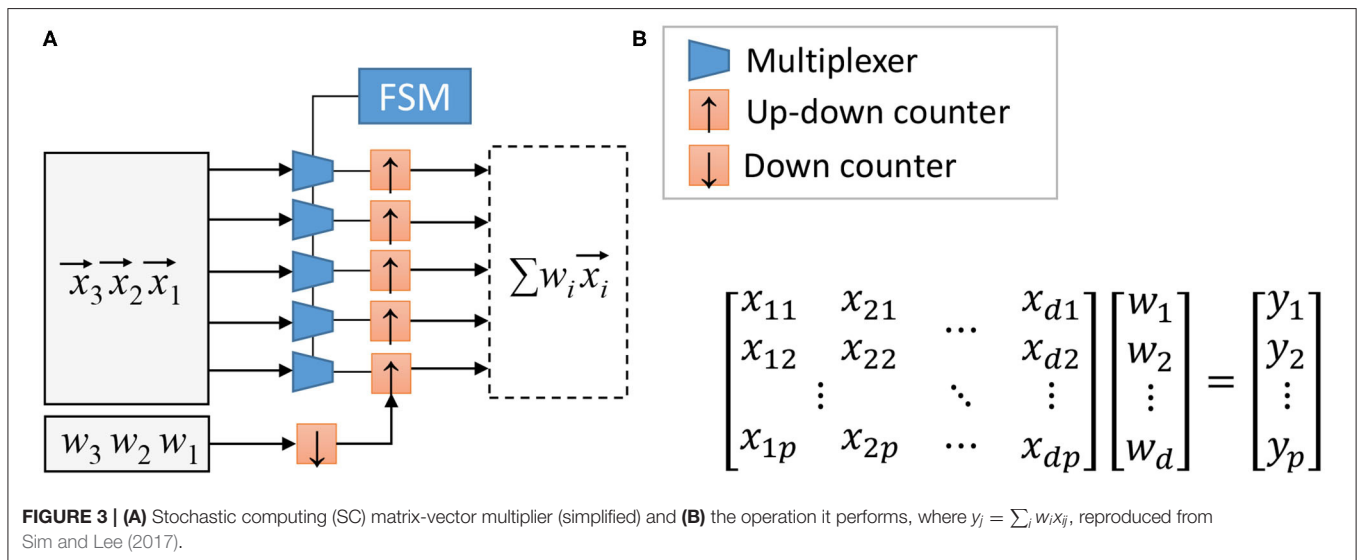
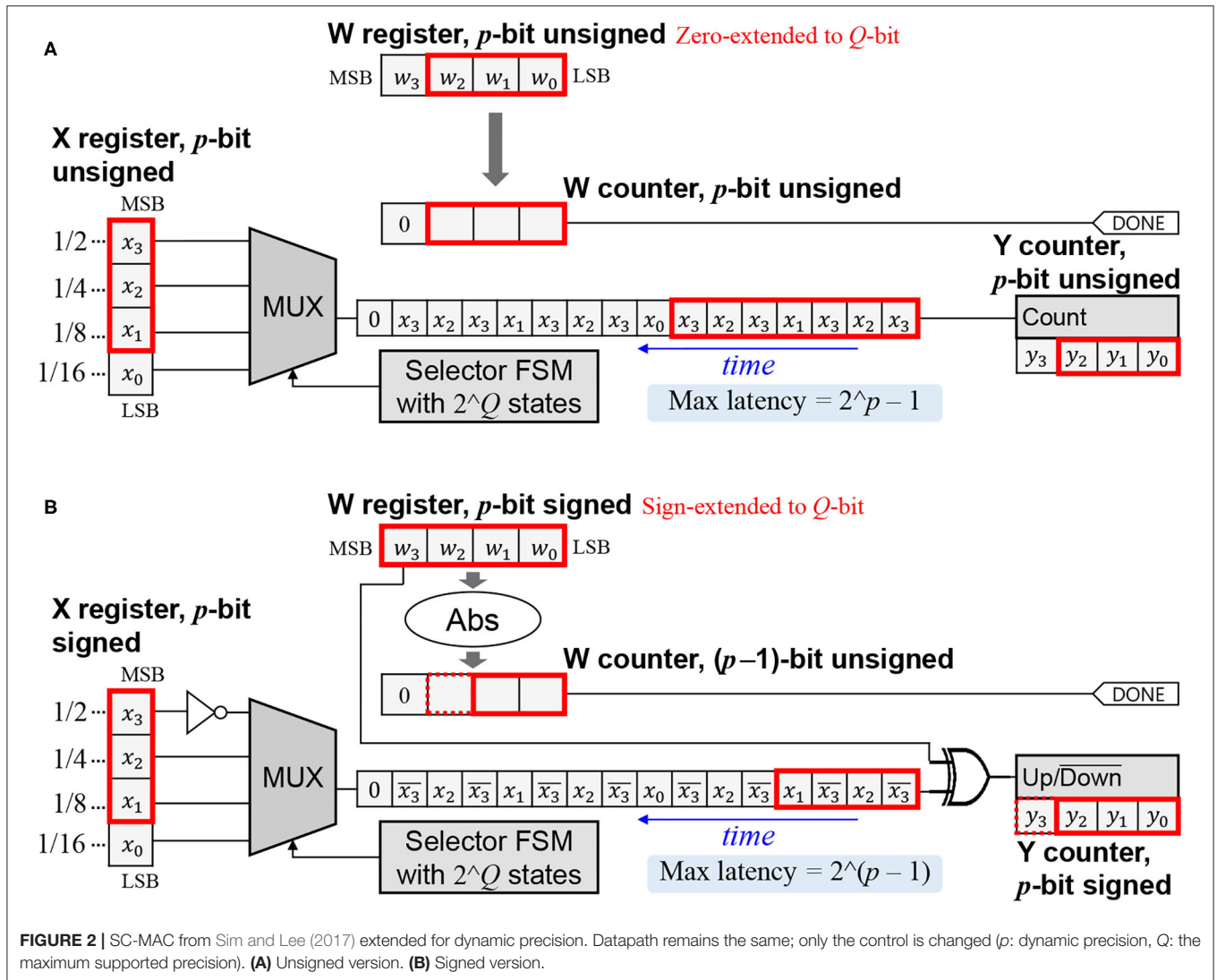
Let us first consider the unsigned version, where the inputs/output are interpreted as fractional numbers between 0 and 1. Let integer $X = x2^Q$ and integer $W = w2^Q$, where Q is the width of the X and W registers. The MUX-FSM circuit is designed to generate a bitstream whose signal probability is close to x (Sim and Lee, 2017). Thus, counting bits from the x -bitstream for W cycles gives approximately $xW = xw2^Q$. Therefore, $y \approx xw$ with Q -bit precision.

In the signed version, the inputs/output are 2's complement numbers between -1 and 1 . Since the MSB (most significant bit) is used as the sign bit, $X = x2^{(Q-1)}$ and $W = w2^{(Q-1)}$. The W counter is initialized to the absolute value of W . If W is positive, feeding the x -bitstream to the Y counter (which is now an up/down counter) for W cycles will give approximately $xW = xw2^{(Q-1)}$. To understand why, note (i) unsigned interpretation of X register after inverting the MSB is $(X + 2^{(Q-1)})/2^Q$, and (ii) the expected contribution of a single bit z to an up/down counter is $(2z - 1)$. Thus, the expected change of Y per cycle is $2(X + 2^{(Q-1)})/2^Q - 1 = x$. If W is negative, the x -bitstream is inverted, resulting in the negated value of $x(-W)$, or xW in the Y counter. In either case, $y \approx xw$ with Q -bit precision (including the sign bit).

3.2. Acceleration of Neural Network

The baseline SC-MAC can be used to accelerate convolution and linear (also known as fully connected) layers (Sim and Lee, 2019). The first question is how to combine multiple MACs to create an array of MACs. **Figure 3** illustrates the matrix-vector multiplier (MVM) block proposed in Sim and Lee (2017), which is to share the weight parameter (w) among the MACs. The j th up/down counter in **Figure 3** accumulates the product terms $x_{ij}w_i$, eventually computing $\sum_i x_{ij}w_i$. In this sense, one SC-MAC, which consists of an MUX and an up/down counter, can be regarded as a synapse-neuron pair.

Constructing the MAC array in this manner has two important benefits. First, since the latency of the SC-MAC is dependent on w , this scheme ensures that all the SC-MACs in an MVM block finishes simultaneously; in other words, there is no synchronization overhead within an MAC array. Second, the FSM and the down counter connected to w can be shared across all MACs within an MVM block, leading to cost-efficient hardware.



To use the MVM array as the main computation engine of a neural network accelerator, we also need to decide the dataflow, or how to parallelize computation. For convolution layers, for instance, computation of the output feature map (OFM) uses the same weight parameters along the width and height dimensions. Hence the convolution computation can be parallelized along the two dimensions as suggested in Sim and Lee (2017), which corresponds to output stationary according to the taxonomy of Chen et al. (2017). To provide the input to the 2D MAC array one can use the input reuse network (Rahman et al., 2016) between the input feature map buffer and the MAC array, or rearrange input data in advance with some duplication as is typically known as im2col (Chetlur et al., 2014). We later extend the 2D parallelism to make a better use of hardware area for neuromorphic applications (see section 4.6).

3.3. Dynamic Precision Scaling Extension

The extension to support DPS on the SC-MAC costs very little hardware. In fact, the datapath remains almost the same except for a few gates, the major difference being in the control logic. Here, we explain the operation of DPS SC-MAC.

The DPS SC-MAC has an additional input p , which is the precision of the input/output values. Figure 2 illustrates an example with $p = 3$ where the maximum precision Q is 4-bit.

Let us first consider the unsigned version, where the operands and the output are interpreted as unsigned numbers between 0 and 1. The output, being accumulated, may grow larger than 1, for which we add extra bits in the Y counter. The X register holds the integer version of x , or $x2^p$, aligned at the MSB. The remaining bits, if any, are not used. The W register is initialized to the integer version of w , or $w2^p$, zero-extended to fill the Q -bit register, i.e., $W = w2^p$. The selector FSM is unchanged regardless of p .

It is easy to see that the latency of multiplication is $W = w2^p$ cycles, which is at most $2^p - 1$. During this period, the LSB (least significant bit) part of X register that is not initialized from x is unused (x_0 in the example). Thus, counting the x -bitstream still approximates xW , with less accuracy due to the reduced precision of W . Since $Y \approx xW = xw2^p$, $y \approx xw$ with p -bit precision.

In the signed version, the X register holds $x2^{(p-1)}$ aligned at MSB. After inverting the MSB, the unsigned interpretation of X register is $0.5 + x/2$ with p -bit precision, assuming a decimal point right before MSB. The W register is initialized to $w2^{(p-1)}$, sign-extended to Q -bit, i.e., $W = w2^{(p-1)}$. If W is positive, the Y counter holds $xW = xw2^{(p-1)}$ after W cycles. If W is negative, it holds $-x(-W) = xW$ due to the XOR gate. In either case, $y \approx xw$ with p -bit precision including the sign bit.

The latency of signed multiplication is $|w|2^{(p-1)}$ cycles. The maximum latency of DPS SC-MAC is $2^{(p-1)}$ when $w = -1$, in which case the W counter requires p -bit, as indicated by the dotted box in Figure 2B (but it never needs more than Q bits). Similarly, the Y counter needs $p+1$ bits, which may exceed Q bits; however, the Y counter has extra bits already in order to serve as an accumulator.

Since the p -bit precision of y always starts from LSB, it means that the decimal point will have to move depending on

the precision. Fixing the decimal point can be done with a single shifter.

3.4. Half-Range Specialization

Since the latency of an SC-MAC is exponential to the precision, saving even 1 bit is very worthwhile. HRS is based on the observation that the range of certain variables, namely, input activations, are guaranteed to be non-negative due to the particular shape of activation function (i.e., ReLU) used in the preceding layer.

One way to exploit the limited range of input is through a data scaling framework as in section 4.3. But data scaling works best for symmetrical ranges, and making asymmetrical ranges symmetrical incurs additional overhead.

Alternatively we can make a full use of input precision in our DPS SC-MAC of Figure 2B by treating x as unsigned. This effectively increases x 's precision to p -bit while the precision of w remains the same [i.e., $(p-1)$ -bit due to 1-bit sign]. Since w is unaffected, latency is also the same. The main effect of this scheme is accuracy improvement: in our evaluation, $(p-1)$ -bit multiplication with HRS shows a similar accuracy as p -bit multiplication without HRS (see Figure 10). Conversely, HRS can achieve a similar accuracy with 1-bit less precision, or at half the latency.

It is important to note that HRS cannot guarantee the same accuracy as that of 1-bit lower precision, since w 's precision is not increased. However, input activation often turns out to require a higher precision than weight parameters Judd et al. (2015), which explains why increasing x 's precision through HRS is very effective in practice. On the other hand, if a layer or network requires higher-precision weight (w) than input (x), the HRS advantage of cost-free increase of x precision by 1 bit will not be very useful, since weight precision is the bottleneck and determines the SC-MAC's precision.

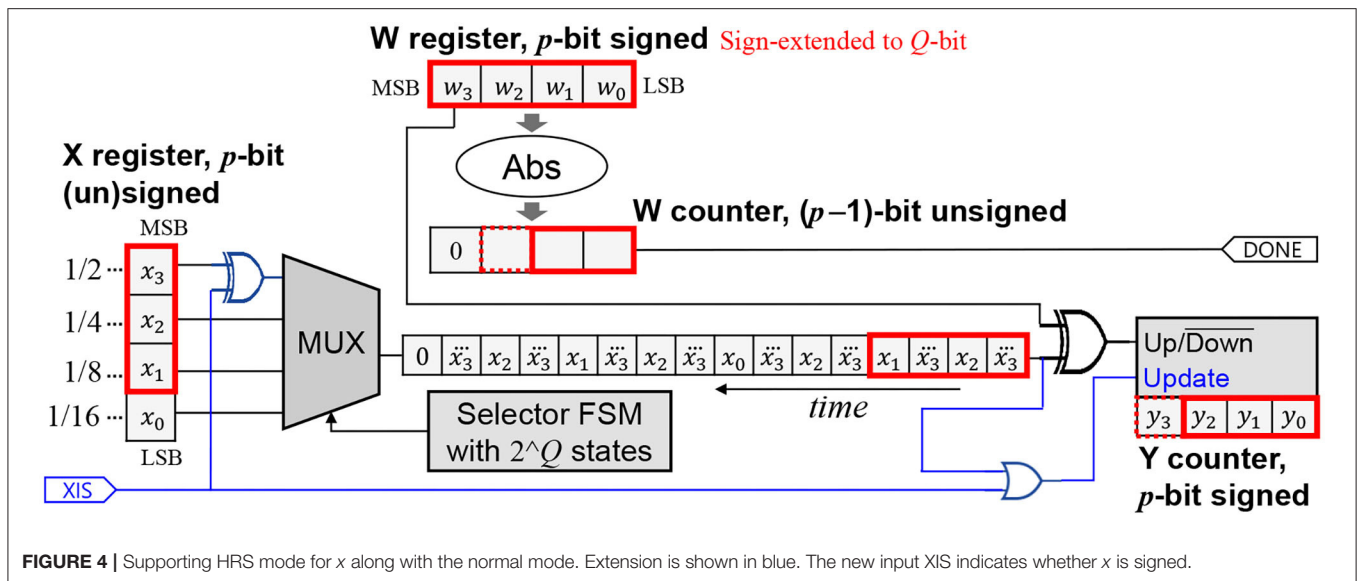
Also important to note is that in order to support layers whose input is not necessarily one-sided (e.g., the first layer), the hardware must retain the original behavior of Figure 2B. Thus, we make HRS runtime-programmable through an extra input XIS (meaning "x is signed"), as shown in Figure 4.

When XIS is 1, the hardware degenerates into the signed version. When XIS is 0, the x part becomes like the unsigned version, and the up/down operation of the Y counter is suppressed if the x -bitstream's output is 0, essentially making it perform either up or down depending on the sign of w , which ensures a correct operation.

4. DESIGN OPTIMIZATIONS FOR DPS SC-CNN

4.1. Hardware Precision vs. Software Precision

So far our notion of precision has been the width of a variable in the application program as represented in conventional digital, which typically ranges up to 32-bit. Quantization is to reduce the precision in the application code. Thus, this kind of precision may be called *software precision*. When converted



into SC, a variable of n -bit software precision requires about a 2^n -bit bitstream.

While a bit-serial multiplier as in (Judd et al., 2016) computes only one bit at a time, it is quite common in SC to employ bit-parallel hardware due to the relative simplicity of an SC multiplier compared to the rest of SC-MAC. To generate a variable of n -bit software precision, a bit-serial SC multiplier needs 2^n cycles, but a k -bit parallel SC multiplier can do it in $2^n/k$ cycles. We define *hardware precision* as the base-2 logarithm of k .

It is straightforward to extend the DPS SC-MAC to a bit-parallel version, which supports integer hardware precisions for efficiency reasons. It is based on the bit-parallel version of the baseline SC-MAC (Sim and Lee, 2017).

4.2. Design Flow

A design objective is to minimize ADP while meeting accuracy constraint, which we set to be 1% point below the reference accuracy achieved by an unquantized version (i.e., floating-point implementation). We consider the following design parameters: (i) data scaling parameters, (ii) software precision of each layer, and (iii) hardware precision of SC-MAC. Next we discuss each of these.

4.3. Determining Data Scaling Parameters

Previous work (Lin et al., 2016) has pointed out the importance of scaling input data to better utilize the limited range of SC or fixed-point representations. The idea is to scale more than covering the worst case input data, such that some of the input values go out of range. It may introduce errors to some input, but those in the range can be represented more precisely. More-than-worst-case scaling is particularly effective when the out-of-range input data get saturated. To avoid the overhead due to scaling, scaling parameters are typically restricted to powers of 2.

The issue here is how to determine data scaling parameters, the effect of which seems highly unpredictable. We use the following scheme.

1. Determine the scaling factor so that all values are within range (i.e., worst-case design).
2. Double the scaling factor and check whether the recognition accuracy improves.
3. Repeat the above while there is improvement.

The above procedure is repeated for each layer, starting from the first layer. We do not retrain the CNN during this procedure. We find this scheme robust as it does not rely on any arbitrary design parameter, which is a major advantage of the scheme. While this algorithm is greedy and not able to address the possible inter-dependence issue among layers, doing so would run into a combinatorial problem, which may require a prohibitive amount of resources for large CNNs.

4.4. Determining Software Precision of Each Layer

Similar to data scaling parameter exploration, here we optimize one layer at a time in order to avoid combinatorial problems. There are also differences. First, precision optimization uses retraining, which is crucial to get meaningful accuracy at low precisions. On the other hand, retraining takes much longer than inference, and can take hours and days for the SC version even when using GP-GPUs for simulation. Second, higher precision is more detrimental than a lower precision can save. Thus, we first find the *uniform* precision for the SC version that satisfies the accuracy constraint with retraining. This can be solved in linear time, since all layers have the same precision. The uniform precision is used as the precision upper-bound for each layer. Third, knowing the uniform precision also helps determine hardware precision (see the next section). Fourth, to speed up the search we use the result of conventional digital implementation's optimized precision. However, since there is usually a gap between the precisions of the two, we use a concept called *precision slack*.

To illustrate *precision slack*, suppose that a conventional digital implementation is optimized to have the following

precisions across five layers: 10–9–5–6–8. Precision slack is the difference in precision between the highest and the current layer, e.g., 0–1–5–4–2 in this example. Then we subtract precision slack from the uniform precision value to get the precision lower-bound. The rationale is that while the SC version gives higher reward for lower precision, its accuracy is also more sensitive to it. Also using very low precisions gives diminishing return (see **Figure 1**) while often hurting accuracy too much. Having a lower-bound makes it easy to do binary search instead of linear search, saving retraining time.

4.5. Determining Hardware Precision

Hardware precision affects both delay and area of our SC-CNN (see section 5.3). Therefore, the decision in this step could affect the optimality of the precision setting found in the previous step as ADP can change as we use a different hardware precision. To avoid this problem, we run this step twice, first for the uniform precision value, then after non-uniform precision setting is found. Finding the best hardware precision is straightforward, and can be done quickly as it has only a linear complexity and does not require retraining (changing hardware precision does not affect recognition accuracy, but only ADP).

4.6. Neuromorphic Optimizations

4.6.1. Motivation

One key difference between neuromorphic vs. deep learning hardware is the separation of computation and memory. In the nervous system, memory is distributed and computation is tightly integrated with memory, whereas in today's deep learning hardware, memory is clearly separated from the compute engine, which can create performance bottleneck for certain applications due to memory wall.

The main reason why today's deep learning architectures use the von Neumann architecture is efficiency. Because digital MACs are large, a typical chip can have only so many of them, which we must use iteratively, or in a time-multiplexed manner, in order to handle large layers and networks. Also because of the large granularity of individual MACs, it would be quite inefficient and difficult in terms of placement and routing if we distribute them among memory blocks.

Contrary to a digital MAC, which consists of an n -bit multiplier and an m -bit accumulator ($m > n$), an SC-MAC is extremely small. It allows us to build a massive array of neurons and synapses on a single chip, for which we explore a much more parallel architecture than the previous SC neural networks.

We also explore a tight integration of memory (e.g., SRAM blocks) with SC-MACs. Even though our SC-MAC takes digital numbers (X and W) as input, only one bit is used at a time (see **Figure 4**). By rearranging the bits in the memory, the MUX can be made redundant, with its function merged into the address decoder of an SRAM block.

4.6.2. More Parallel Architecture for SC-CNN

While the size of our SC-MAC depends on the hardware precision as explained in section 4.1, it can be up to 63.9 times smaller than a digital MAC. To better utilize the massive number of SC-MACs available, we parallelize along all dimensions of the

convolution kernel. **Figure 5** lists the C code of a convolution kernel that is tiled along all four dimensions of Z, M, R, C . The remaining two loop levels not tiled, K_r, K_c , are typically very small. Those tiled loops are unrolled in hardware, meaning that the MAC array consists of $T_Z \times T_M \times T_R \times T_C$ SC-MACs, and is able to perform the same number of SC-MAC operations every cycle. This is in contrast with the previous SC-CNNs (Sim and Lee, 2017; Sim et al., 2018), where only R, C dimensions are unrolled, thus having saturating efficiency when the array size increases (see section 5.6). Conventional digital accelerators also, such as Chen et al. (2014, 2017), not having been optimized for such a high degree of parallelism, suffers the same limited efficiency issue.

Parallelizing along the M -loop is similar to parallelizing along the R - or C -loop because M, R , and C are all output feature map dimensions; we replicate the 2D MAC array T_M times. But there is a downside. Each of the 2D MAC arrays uses one weight value per cycle, hence there are T_M weights overall that need to be supplied per cycle. Consequently, the T_M MAC arrays may have different latency values, which can incur synchronization overhead among the 2D MAC arrays.

The main idea of parallelizing along the Z -loop is to increase the number of inputs for the up/down counter in **Figure 4** by T_Z times. Instead of having a single bitstream, we now have T_Z bitstreams coming from different input channels; thus, we employ a T_Z bitcount logic to combine T_Z bits into an integer, which is then accumulated. In fact, the up/down counter in **Figure 4** has three operations, i.e., up, down, and no-op (when update is zero), due to the HRS optimization. Hence, the input bitstreams are ternary, and the bitcount logic is extended to handle ternary input. Similar to the parallelization along the M -loop, the weight parameters from the T_Z input channels may all be different. There are T_Z down counters corresponding to the T_Z weights. The done signal from a down counter forces the corresponding ternary input to zero, which makes the input effectively ignored by the bitcount logic.

4.6.3. Tight Integration of SRAM and SC-MAC

Among the main components of an SC-MAC, i.e., MUX and up/down counter, the MUX can be made redundant if we rearrange the input data in the SRAM. Suppose input x is 8-bit. In the conventional memory storage, each byte of the input SRAM contains one value of x , the next byte containing the next x , and so on. In the previous work (Sim and Lee, 2017; Sim et al., 2018), all these values of x are loaded simultaneously into the input registers, but only one bit is accessed per cycle through the selector FSM.

More specifically, let x_1, x_2 , etc. be 8-bit values loaded to the input registers of an MAC array. In other words, x_j is the j th element of the input vector \vec{x}_i in **Figure 3A**. Let $x_j^{(k)}$ be the bit k of x_j where $0 \leq k \leq 7$. Then in the first cycle we need a set of bits, $x_1^{(7)}, x_2^{(7)}$, etc., and in the next cycle we need another set of bits, $x_1^{(6)}, x_2^{(6)}$, etc. This scheme may be called *bit-major*.

Now we propose to store the input data in such a way that the bits needed together are stored together in the same byte as much as possible. For instance in the above example, bits

```

for ( $m_1 = 0$ ;  $m_1 < M$ ;  $m_1 += T_M$ )
  for ( $r_1 = 0$ ;  $r_1 < R$ ;  $r_1 += T_R$ )
    for ( $c_1 = 0$ ;  $c_1 < C$ ;  $c_1 += T_C$ )
      for ( $z_1 = 0$ ;  $z_1 < Z$ ;  $z_1 += T_Z$ )
        for ( $i = 0$ ;  $i < K_r$ ;  $i++$ )
          for ( $j = 0$ ;  $j < K_c$ ;  $j++$ )
            for ( $m = m_1$ ;  $m < \min(M, m_1 + T_M)$ ;  $m++$ )
              for ( $r = r_1$ ;  $r < \min(R, r_1 + T_R)$ ;  $r++$ )
                for ( $c = c_1$ ;  $c < \min(C, c_1 + T_C)$ ;  $c++$ )
                  for ( $z = z_1$ ;  $z < \min(Z, z_1 + T_Z)$ ;  $z++$ )
                     $B[m][r][c] += W[m][z][i][j] \times A[z][Sr + i][Sc + j];$ 

```

Symbol	Meaning
M	# of output channels
R	# of rows in OFM
C	# of columns in OFM
Z	# of input channels
K_r, K_c	size of 2D conv. filter
S	stride of 2D conv.

FIGURE 5 | Convolution layer tiled along four loop levels (M, R, C, Z). Arrays A , B , and W are input feature map (IFM), output feature map (OFM), and weight parameters, respectively. The four innermost loops are hardware-unrolled, creating a 4D-parallel architecture.

$x_1^{(7)}, x_2^{(7)}, \dots, x_8^{(7)}$ will make up one byte, and $x_1^{(6)}, x_2^{(6)}, \dots, x_8^{(6)}$ will make up another byte. The latter byte may never need to be accessed simultaneously as the former, hence can be stored in the depth direction of an SRAM block. We call this scheme *lane-major*. Using the lane-major scheme can eliminate both input MUXes and input registers. The scheme can also reduce the input memory bandwidth, or the maximum number of bytes we read from the input memory in a cycle, by N times compared with bit-major, where N is the width of input in bytes times 8. For instance, if the input data are 12-bit wide, it still requires 2 bytes, thus the saving is 16 times. Using the lane-major scheme does not significantly affect the capacity of the input memory needed but does affect the aspect ratio; we now need deeper memory.

On the other hand, since we have eliminated input registers, we must access the input SRAM every cycle, potentially increasing the energy consumption. In the worst case, an 8-bit input data may need to be accessed 256 times in the lane-major scheme. However, the actual number of accesses depends on the weight value, which is typically small. Also when using dynamic precision scaling, the actual precision at the moment can be much less than the width of the input data stored in the memory. Thus, our scheme fits well with DPS.

While the input data are stored as lane-major, the output data as produced in an SC-MAC follows bit-major. Therefore, we need to convert the bit packing scheme of the output data. This can be done when the output data are loaded from the external memory to the on-chip input buffer, at which time we also apply the im2col transformation (Chetlur et al., 2014). In addition, we can optimize away the XOR gate required by HRS, by doing the MSB flipping in advance. It can be done during the bit packing conversion.

5. EXPERIMENTS

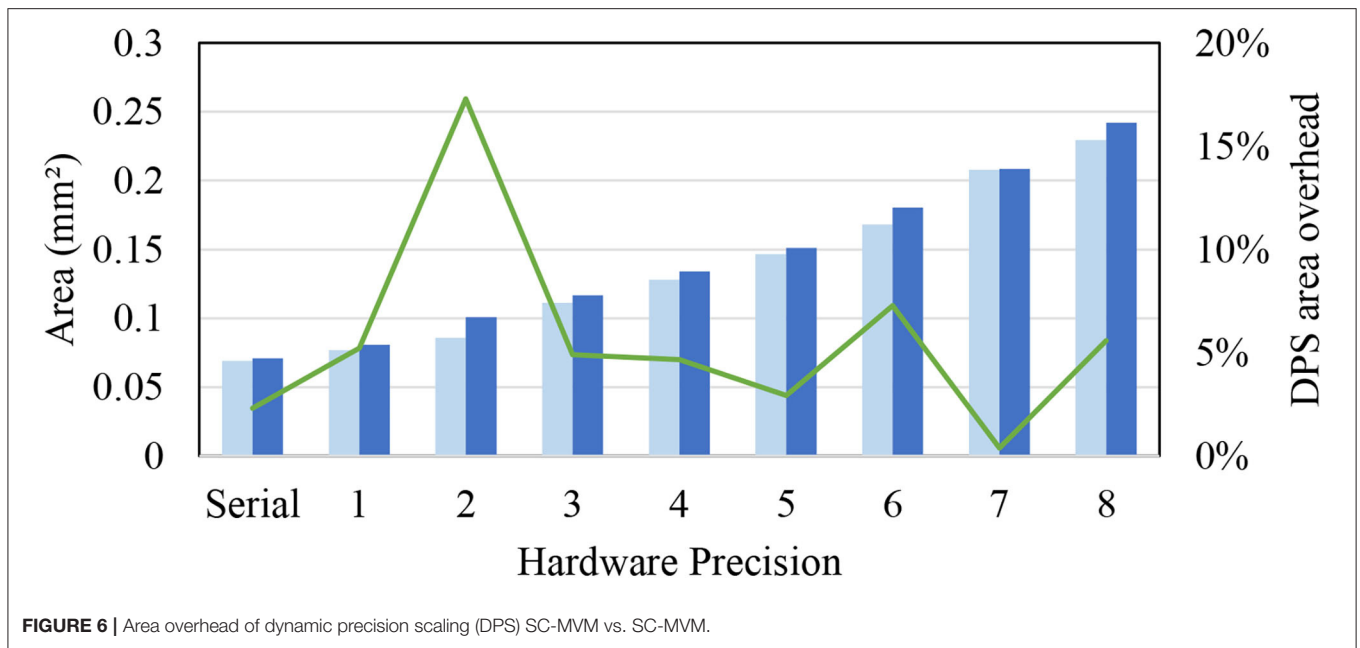
5.1. Experimental Setup

To evaluate our approach, we use CNNs targeting ILSVRC2012 (ImageNet Large Scale Visual Recognition Challenge 2012), such as AlexNet, VGG, and GoogLeNet, in addition to smaller ones.

For training and accuracy evaluation, we use Caffe (Jia et al., 2014) extended to model the functionality of DPS SC-MAC. For retraining (also called fine-tuning), 5,000 update iterations were performed starting from the reference models of the Caffe Model Zoo.² For learning parameters, such as weight decay and batch size, we use the same values as used in the reference solver script provided with the model, with the only exception of the base learning rate, which is scaled down by 10× from that of the reference script. During retraining, forward propagation is done using the SC algorithm but back-propagation is done using floating-point arithmetic with weight update done to real-valued weights, which is essentially the same procedure used in training quantized neural networks (Hubara et al., 2017). Note that retraining is needed not only for SC-DNNs but digital DNNs also, and that the computational overhead of retraining is very little compared with that of the baseline training (5,000 vs. 450,000 iterations in the case of AlexNet). Recognition accuracy is reported for the first 10,000 images (out of 50,000) of the ImageNet validation set. The SC-CNN architecture is modeled cycle-accurately to generate exact cycle counts in a data-dependent manner.

We have extended the SC-MVM (matrix-vector multiplier) (Sim and Lee, 2017) to support our DPS SC-MACs, which is referred to as DPS SC-MVM. Our DPS SC-MACs have a few variants depending on hardware precision. Our DPS-2^p processes 2^p bits per cycle, therefore being roughly equivalent to p -bit parallel digital logic. We have implemented the previous SC-MVM (Sim and Lee, 2017), our DPS SC-MVM, and the conventional digital baseline MVM in Verilog and synthesized them using Synopsys Design Compiler. All syntheses were done for the same target frequency of 1 GHz, although SC is likely to meet higher frequency. The conventional digital baseline uses fixed-point binary multipliers with rounding accumulators. The area for Stripe (Judd et al., 2016) is estimated to be 207% of the digital baseline as per the paper, which however does not provide power result. Only convolution layers are

²<https://github.com/BVLC/caffe/wiki/Model-Zoo>



accelerated in all the approaches compared, permitting us to use ideal convolution layer speedup for delay comparison. Maximal accuracy degradation is set to 1% point.

Our main figure of merit is area-delay product (ADP), which is the product of MVM area and average MAC cycles (thus the lower, the better), or its inverse representing operations per area. ADP is chosen because in addition to being a meaningful area-efficiency metric, it permits direct comparisons with previous work (e.g., Judd et al., 2016). Though we also report power and energy results for our designs, these metrics tend to vary a lot, affected more by such factors as on-chip and off-chip memory accesses (Chen et al., 2014), which are not the main focus of this paper, than the design of processing element (PE) arrays.

5.2. Area Overhead of our DPS SC-CNN

Figure 6 compares the area of our proposed DPS SC-MVM against the previous SC-MVM (Sim and Lee, 2017). The DPS SC-MVM includes our optimizations, such as HRS, which have very small extra logic (see **Figure 4**). Not surprisingly, the graph shows that the area overhead of ours is mostly small, typically at around 5%, though varied depending on the hardware precision shown on the *x*-axis. The graph also shows that the area is linearly proportional to the hardware precision, or logarithmically to bit-parallelism. This is due to the optimization exploiting the structure of the bitstream ordering. Overall the average area overhead is 6%, which is small.

5.3. Effect of Software and Hardware Precision on ADP

Figure 7 shows the ADP trend as we vary software precision. For our ADP result, we use AlexNet parameters as our SC-MVM has data-dependent variable latency. The digital baseline does not support dynamic precision, thus has constant ADP. For Stripe, delay is proportional to the precision, resulting in linear

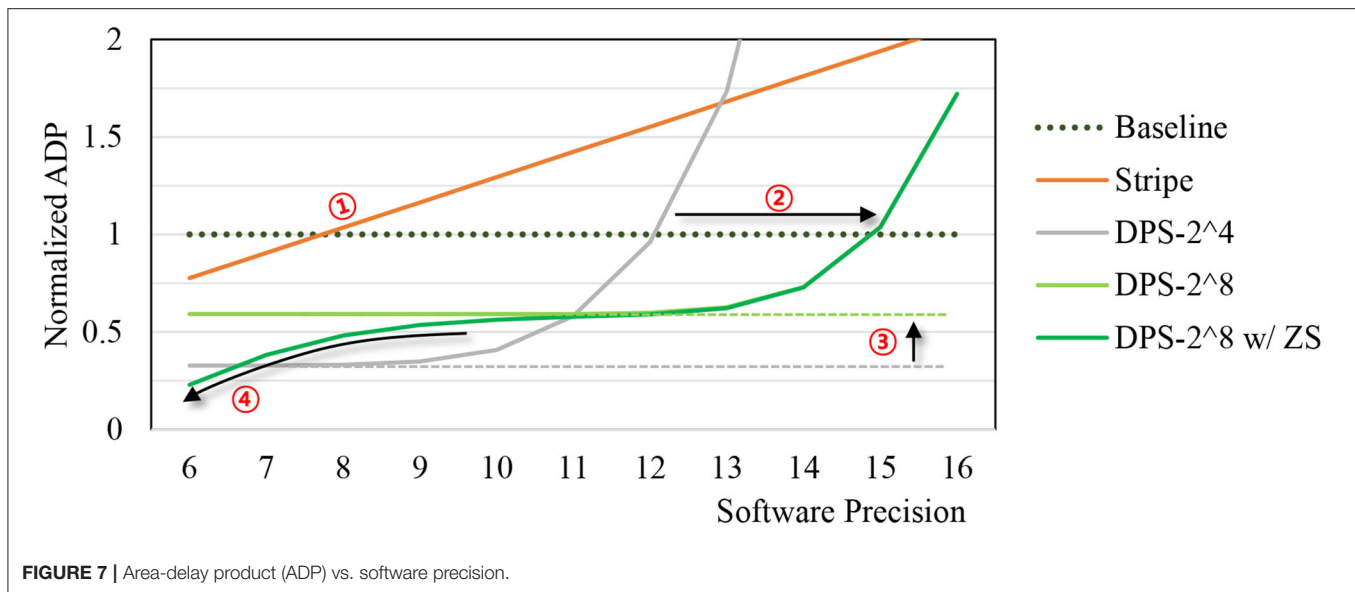
ADP. The graph shows that Stripe becomes inefficient over the conventional digital baseline beyond 7- or 8-bit (①). DPS-2⁴, which is our DPS SC-MVM with hardware precision of 4, shows exponentially increasing ADP as software precision increases. But the range of software precision for which DPS-2⁴ is more efficient than conventional digital is wider than that of Stripe.

DPS-2⁸, which is our DPS SC-MAC with hardware precision of 8, can widen the efficient operating range even further (②). At the same time, it has higher ADP than DPS-2⁴ when software precision is lower (③), as it is more optimized for higher precision workload. Some of the efficiency loss can be reclaimed by zero skipping (④), which is to skip computation of multiplication whose weight operand is 0 (after quantization) as shown in the graph.

As demonstrated previously, ADP depends on hardware precision. **Figure 8** shows ADP vs. hardware precision (a) for a CNN (AlexNet) and (b) for multiple CNNs. As hardware precision increases, the average delay decreases until it reaches saturation, whereas area increases more or less linearly to hardware precision. This suggests that there is an optimal hardware precision to minimize ADP. In the case of AlexNet in **Figure 8A**, for instance, ADP is minimized at hardware precision of 4, or DPS-2⁴. But in other CNNs, different points can be optimal as there are different weight distributions and precision requirements depending on the CNN. **Figure 8B** shows how ADP as well as optimal hardware precision changes depending on application. Understandably, large and complex CNNs seem to be better off with higher hardware precision. Our hardware precision for the multi-application scenario (see the next section) is chosen based on this profile.

5.4. Multi-Application Scenario

Figure 9A compares our DPS SC-CNN and previous CNN implementations. First, ours is highly area efficient, which is not



surprising given the area efficiency of SC. Since Stripe is bit-serial, we use 16 times as many MACs for Stripe as in the baseline, so that the two can have the same throughput in the worst case (i.e., when the CNN's software precision is 16-bit). Hence, Stripe has the largest tile area as shown in **Figure 9B** due to the large number of MACs it has; the others have 256 MACs only.

Figure 9C shows average MAC cycles and **Figure 9D** shows the ADP result, which is normalized to the digital baseline. First, all these results are from implementations that achieve <1% point accuracy drop from the reference floating-point implementations (see **Table 1**). That SC-CNNs can achieve this high accuracy for large CNNs is very significant. Also this is why this graph has no comparison with previous SC-CNNs. Second, at the same time the efficiency of ours as measured in ADP is actually higher than that of conventional digital, often significantly. For instance, DPS-2⁸, which is optimized for large CNNs, shows consistently better results than the conventional digital designs. It also demonstrates the flexibility as well as efficiency of our DPS SC-CNN. Third, the optimal design as measured in geometric mean of ADP is DPS-2⁶ for this mix of CNNs, which is obviously influenced by the existence of a small network. But our scheme can flexibly support different workloads through the hardware precision, while simultaneously being able to support dynamic software precision at runtime. **Figures 9E,F** present the power and energy comparisons, which show very similar trends as those of the area and ADP comparisons in the same figure. Overall, our DPS-2⁶ can achieve over 2× and 1.5× improvements compared with the baseline and Strip, respectively, in terms of operations per area.

5.5. Single Application Comparison

We also compare different implementations including the previous state-of-the-art SC-CNN (Sim and Lee, 2017) for a single application scenario, i.e., when we design and use a chip for just a single CNN. We use AlexNet as the target CNN. **Figure 10** shows area, average delay, and ADP results in one

graph, all normalized to that of the digital baseline. For SC designs, hardware precision is set to 4. Maximum software precision supported (Q) is determined to be the minimum value that meets the recognition accuracy constraint, which is largely dependent on how accurate the MAC is. The digital baseline requires 9-bit while the previous SC-CNN requires 11-bit. Our DPS SC-CNN requires 10-bit with uniform precision; dynamic precision setting is listed in **Table 1**.

The graph shows that the previous SC-CNN has smaller area than the digital baseline but its average delay is much higher, which is attributed to the high precision requirement. Applying HRS to it (but not DPS) can reduce precision requirement by 1-bit with significant saving in delay, but its average delay is still higher than that of conventional digital. Applying DPS further gives 14% reduction in ADP, achieving the best efficiency. The relatively weak impact of DPS is due to the small number of layers in AlexNet and our limited precision exploration. For deeper networks and if we can use the optimal precision combination, the impact could be higher. Even with these limitations our proposed design achieves 34 and 46% reduction in ADP (or 52 and 85% increase in operations per area) over the digital baseline and the previous SC-CNN, respectively.

5.6. Efficiency of Neuromorphic Architecture

Thanks to its extremely small size, SC-MAC allows us to explore more flexible architectures including significantly more parallel architectures and tight integration of memories with compute elements. In one of those architectures, which more closely resembles the nervous system and therefore is referred to as *neuromorphic architecture*, we use the following parameters.

- It is bit-serial (hardware precision $p = 0$), meaning that each SC-MAC processes only one bit in a cycle.
- The tiling parameters (see **Figure 5**) are as follows: $T_Z = T_M = T_R = T_C = 16$.

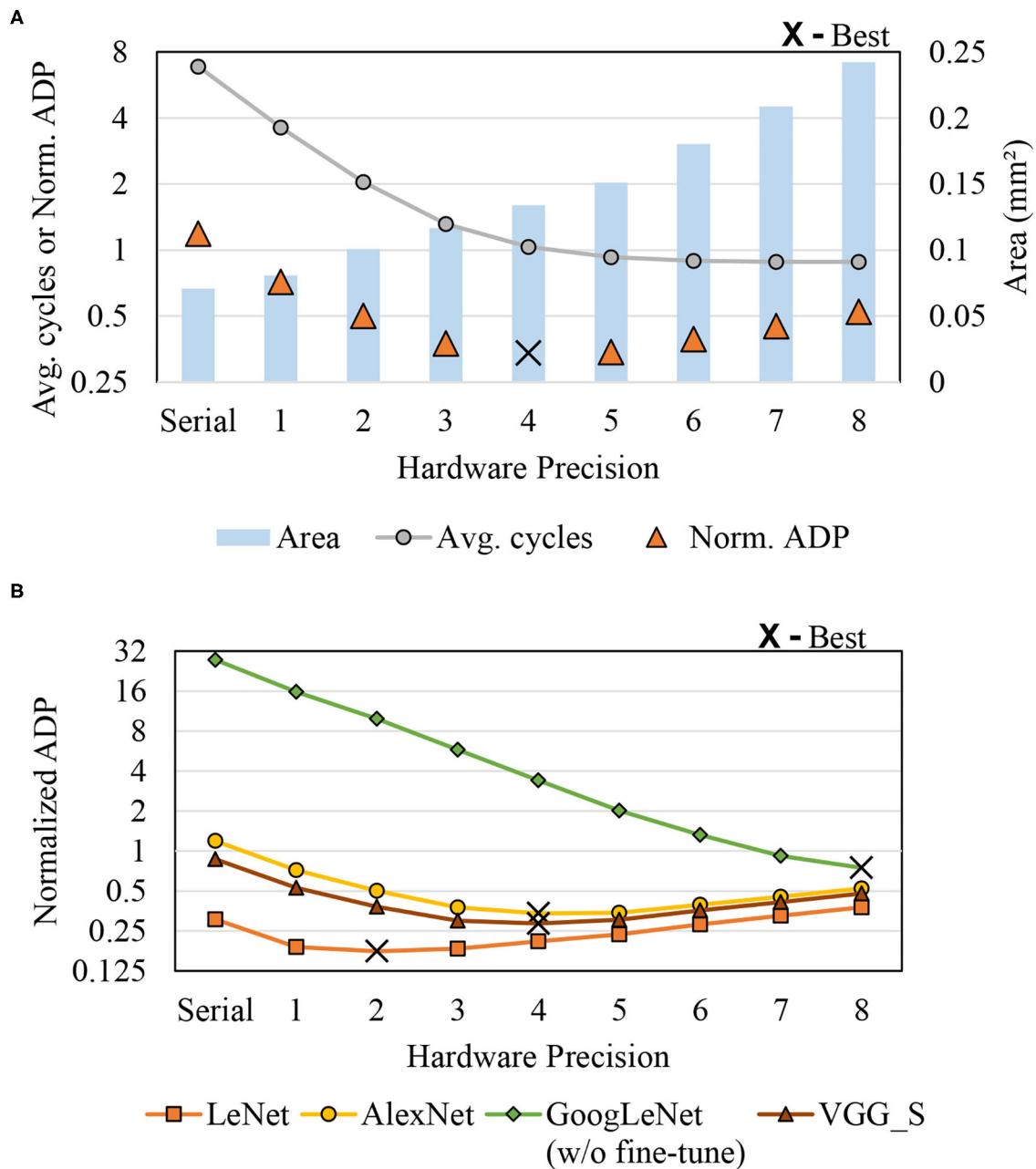


FIGURE 8 | Area-delay product (ADP) vs. hardware precision. **(A)** AlexNet and **(B)** Multi-application.

- Input SRAM is directly integrated into SC-MAC as described in section 4.6.3.

The same synthesis setting is used as described in section 5.1 including the target clock frequency.

We compare three cases: digital, DPS SC-CNN (DPS-2⁴), and the neuromorphic architecture. The result is summarized in **Table 2**. For fair comparison, we use the same number of synaptic connections, which is set to 64K (2¹⁶). This means that all the three architectures compared here have the same number

of multipliers or their equivalents. In the table, the first two architectures, Digital and DPS-2⁴, are the same as in **Figure 9A**, with only 256 MACs or synaptic connections, but added here for comparison. CNN cycles of MNIST for digital and DPS-2⁴ are equivalent because the network precision requirement is 4 (excluding sign-bit) and DPS-2⁴ can process the maximum length of stochastic stream at a cycle. Their “large” versions are created by increasing the tiling parameters; each has two tiling parameters, which are multiplied by 16 each. The ADP column is the geometric mean for both networks.

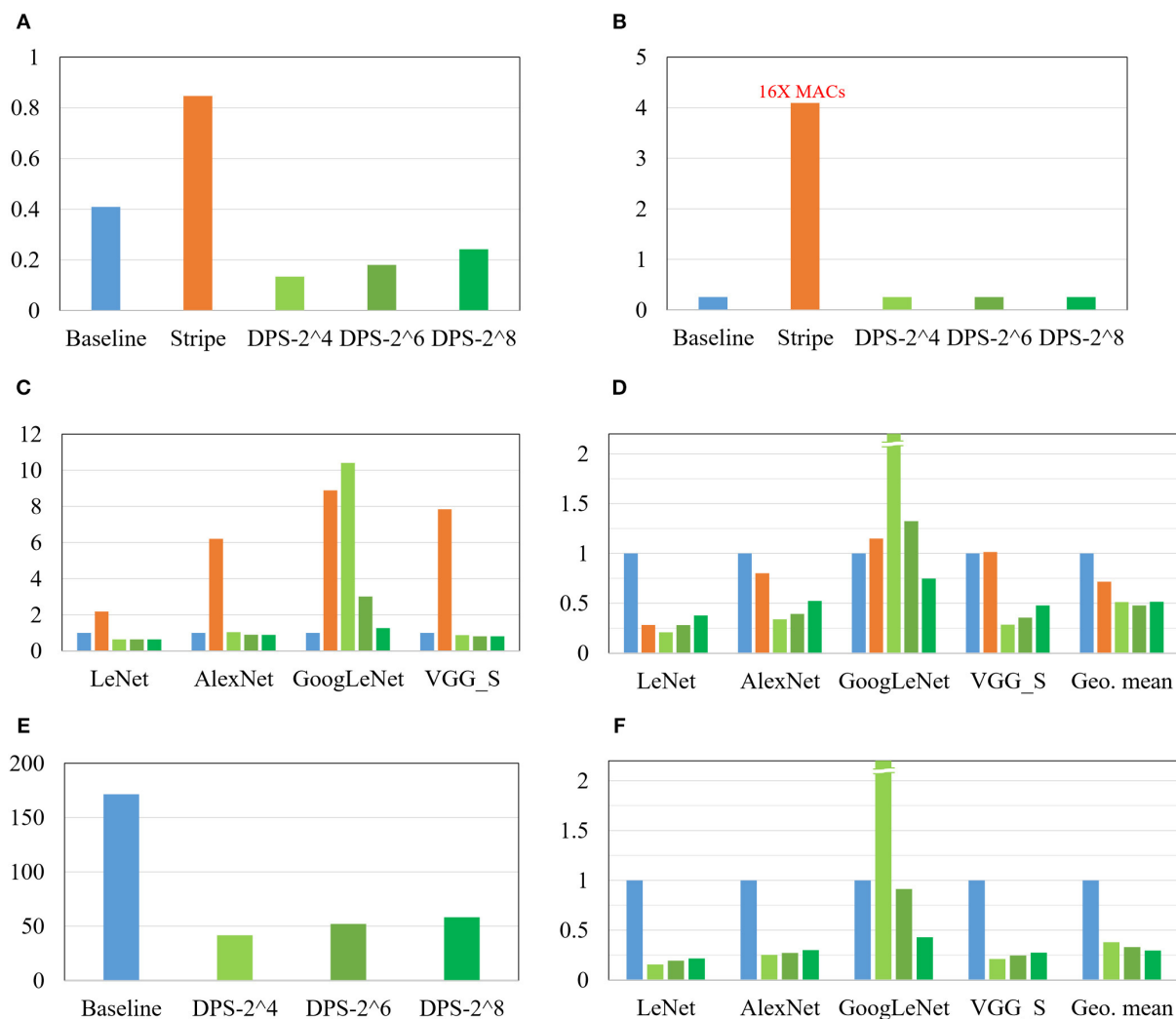


FIGURE 9 | Comparison with the digital baseline and Stripe. **(A)** Area (mm²), **(B)** #MAC of a tile (1 k), **(C)** Average cycles, **(D)** Normalized ADP, **(E)** Power (mW), and **(F)** Normalized energy.

TABLE 1 | Recognition accuracy (for 10K images) and dynamic precision scaling (DPS) precision setting.

CNN	Baseline accuracy (float)	DPS accuracy	DPS precisions found
MNIST	0.9904	0.9826	5 (uniform)
AlexNet (top-5)	0.8070	0.7999	10-9-8-9-9
GoogLeNet (top-5)	0.8926	0.8844	13 (uniform, w/o fine-tuning)
VGG_S (top-5)	0.8341	0.8247	9-9-10-9-10

The table suggests that among the three architectures with 64K synaptic connections, the neuromorphic architecture has the best area, performance, and ADP. In terms of area, the neuromorphic architecture shows several dozen times higher density, thanks to the use of bit-serial SC-MAC and input MUX elimination (enabled by SRAM integration). Yet, its

latency is actually lower than that of the others. The DPS SC-CNN architecture suffers extremely low utilization, which is the main culprit of the architecture when the number of MACs is very high. Simply it is not designed to be very scalable, which is addressed in the neuromorphic architecture. While the neuromorphic architecture has its own weakness, i.e., synchronization overhead, it is relatively mild. Our simulation result shows that the overhead increases average MAC latency by about 2.96 and 6.97 times for MNIST and AlexNet, respectively, compared with when the synchronization overhead is ignored. As a result, the neuromorphic architecture achieves orders of magnitude improvement in ADP over DPS SC-CNN.

The table also provides comparisons with previous digital DNN accelerators. DianNao (Chen et al., 2014) is similar to our *Digital* implementation employing the same number of MAC units but based on a different dataflow, as a result of which it has lower throughput than our digital implementation.

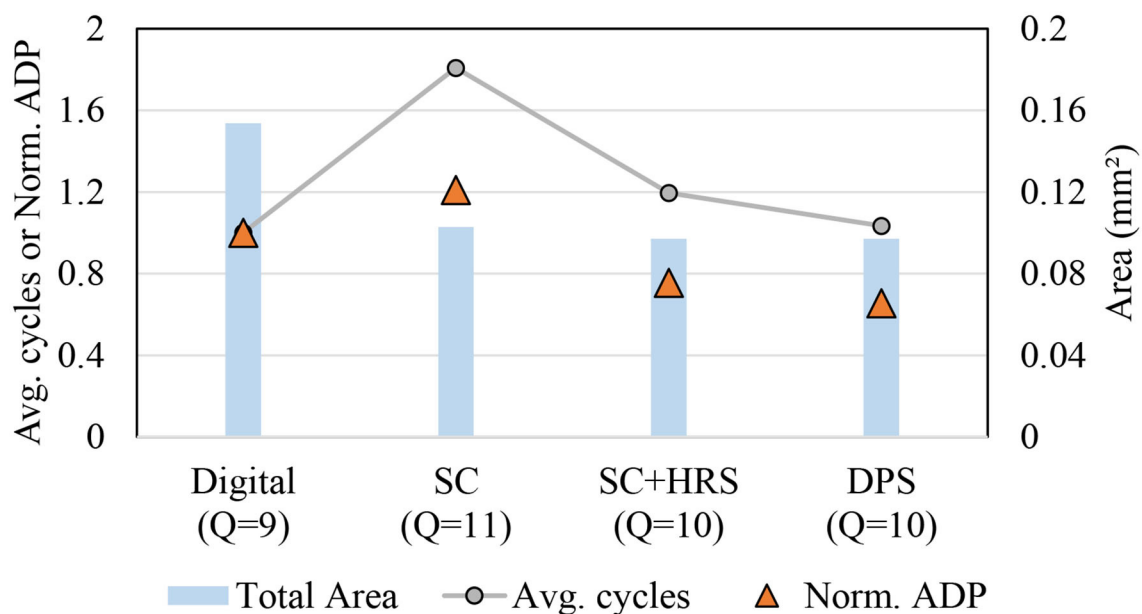


FIGURE 10 | Comparison with previous SC-CNN on AlexNet.

TABLE 2 | Processing element (PE) array comparison.

	Architecture (PEs)	#Synaptic connections	Area (mm ²)	#Synaptic connections/Area	CNN cycles		Area-delay product (norm.)	Power (mW)	Energy (norm.)
					MNIST	ImageNet			
Small	Digital	256	0.41	630	27.0K	3.8M	1	171.42	1
	DPS-2 ⁴	256	0.13	1,985	27.0K	4.3M	0.34	41.62	0.26
Large	Digital large [†]	64K	104.04	630	25.5K	2.3M	195.52	43884.54	195.52
	DPS-2 ⁴ large [†]	64K	33.02	1,985	25.5K	2.4M	62.73	10654.98	47.98
	Neuromorphic	64K	1.63	40,261	2.3K	1.1M	0.65	489.26	0.46
Small	DianNao*	256	0.85	302	41.6K	4.4M	2.79	132	1.03
	Eyeriss*	168	9.63	17	–	20.7M	–	–	–

Our neuromorphic PE array shows much higher synaptic density and better scalability than the scaled-up versions of Digital (the baseline) and our DPS-2⁴. The Digital and DPS-2⁴ PE arrays share the same architecture. For context, we also compare the baseline architecture with two previous accelerator architectures (*based on published papers, [†]estimated).

The area number of DianNao is after place-and-route, and thus includes metal wiring space as well, whereas those of our designs are based on synthesized logic gates only. But even after discounting the differences due to methodology, the compute density (i.e., synaptic connections per area) of DianNao is extremely low, making it unsuitable for neuromorphic architectures, where a large number of non-time-multiplexing neurons are expected. We note that the same weakness plagues our digital implementations as well. Eyeriss (Chen et al., 2017) is a well-known systolic array architecture to accelerate CNNs. While it is one of the most energy-efficient digital CNN accelerators, it has the lowest compute density (synaptic connections per area) in our comparison. That is because Eyeriss employs large PEs as well as many intra-PE registers and complex inter-PE connections, making it hard to scale to tens of thousands of PEs.

When the number of MACs is small, or when there is no area constraint, the DPS SC-CNN architecture achieves the best ADP, beating the neuromorphic architecture by about 2×. The neuromorphic architecture, on the other hand, shows very competitive ADP at a high throughput level. Note that the neuromorphic architecture has the exactly the same accuracy as the DPS SC-CNN, whose accuracy drop is <1% as shown in Table 1. All in all, these results indicate that our SC-based neural network is very flexible and scalable to accommodate various applications as well as area constraints.

5.7. Fault Tolerance

To evaluate the fault tolerance of our proposed schemes, we have performed an error injection experiment. For the fault model, we assume that random bit flip can occur at the input, as is done in a previous study on SC (Qian et al., 2011). The SRAM memories

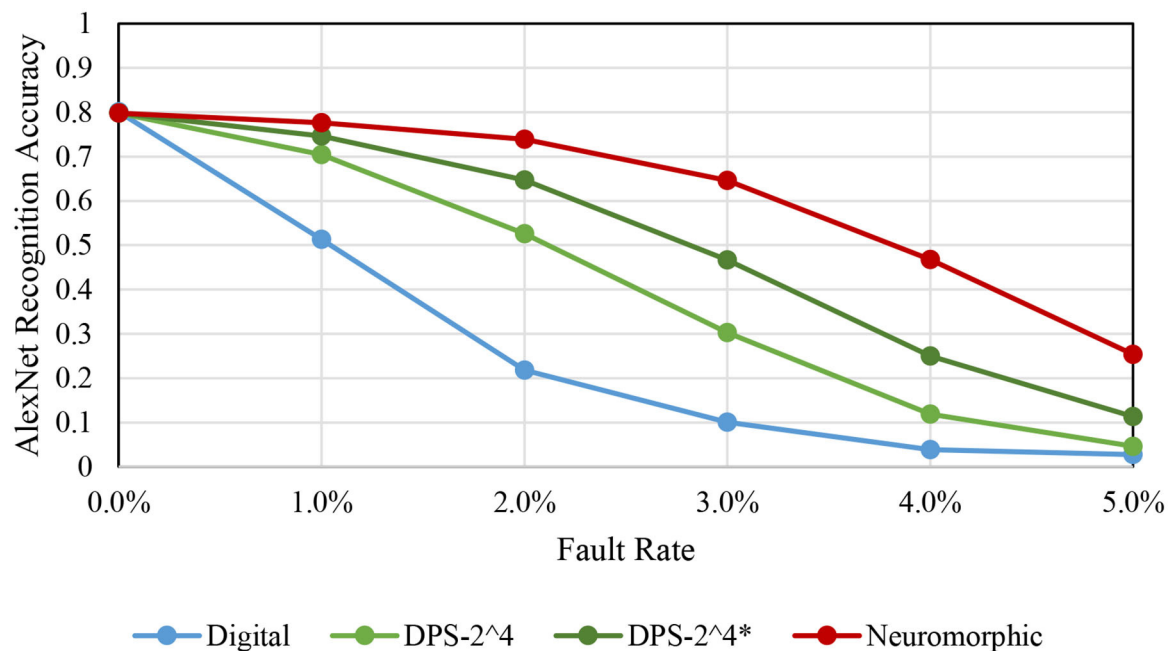


FIGURE 11 | Fault tolerance comparison among different schemes, for AlexNet.

TABLE 3 | Feature comparison (O: supported, X: not supported).

Feature	Digital			SC			Ours
	DianNao (Chen et al., 2014)	Judd et al., 2016	Eyeriss (Chen et al., 2017)	Kim et al., 2016	Sim et al., 2017	Sim and Lee, 2017	
Large (≥ 5 Conv. layers) CNNs	O	O	O	X	X	X	O
Tile based	O	O	O	X	O	O	O
Per-CNN precision	X	O	X	X	X	X	O
Per-layer precision	X	O	X	X	X	X	O
Per-bit precision	X	O	X	O	O	O	O
Multi-bit acceleration	X	X	X	X	X	O	O
Variable latency operation	X	X	X	O	X	O	O
Half-range specialization	X	X	X	X	X	X	O

are assumed to be protected, such as using hardened logic or ECC (error correcting code). For a given fault rate f , we flip the bits of input registers, whose size varies depending on the scheme, with the same probability f . This fault model is integrated into the Caffe framework. No retraining is performed, but only inference, in the presence of faults.

Figure 11 shows accuracy degradation for AlexNet as we vary fault rate. First, we observe that SC-based implementations show significantly higher fault tolerance than the conventional digital implementation, which agrees with previous studies (Qian et al., 2011; Zhakatayev et al., 2018). Second, there is quite a variance among the SC-based implementations. The neuromorphic implementation, which is based on bit-serial SC, shows the highest fault tolerance whereas the bit-parallel version, DPS-2⁴, is less error resilient. There are a number of differences

between the two architectures. One relevant fact is that the bit-parallel version performs a weighted bitcount operation to process multiple bits in parallel, which is more like digital logic than SC, and thus may contribute to its lower fault tolerance.

Another difference is that the neuromorphic architecture reloads input every cycle due to the tight SRAM integration. (The neuromorphic architecture has input registers too like the other architectures.) To test if this contributes to the higher fault tolerance, we test a variant of the DPS-2⁴ scheme, denoted by DPS-2⁴*, which is to reload input every cycle even when it is not necessary to do so. Our experimental result in Figure 11 clearly shows that input reloading helps. This may come as a surprise, but while input reloading does not lower the average number of faults in the circuit, it does lower the chance of having *correlated faults*, faults that occur at the same bit position of

the input and therefore are more detrimental to the correctness of computation. The neuromorphic architecture has the least correlated faults, which helps achieve the highest fault tolerance.

5.8. Feature Comparison With Previous Work

In addition to performance numbers, our solution proposed in this paper has many important features as summarized in Table 3. The key factors that make our work much more efficient and accurate than the previous work are the combination of variable latency, dynamic precision (i.e., per-layer precision), multi-bit acceleration (crucial for larger CNNs), and HRS (which is specific to SC). In addition, ours has high fault tolerance inherent with SC.

6. CONCLUSION

In this paper, we presented a bitstream-based neural network, which is a highly optimized and deterministic version of SC neural network. Thanks to many optimizations including dynamic precision scaling and half-range specialization in addition to the fundamental redesign of the SC multiplication operation, our SC-CNN can achieve both very high accuracy and high efficiency up to ImageNet-targeting CNNs. The SC-CNN owes some of its accuracy advantage to deep learning training algorithms, such as backpropagation. However, it has a distinct set of advantages over deep learning models due to SC, such as precision flexibility and error resilience. These advantages can be very useful, for instance, when designing a single piece of hardware that needs to efficiently support various neural networks with different precision requirements and when computation may not be reliable due to advanced semiconductor process scaling. The flexibility of precision comes with the challenge of optimizing it. Currently, our optimization flow is

greedy and slow due to the retraining of SC-CNN. Finding better methods to determine optimal precision settings more quickly remains for future work.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

HS did the design, implementation, and experimentation in the paper, and also made the initial draft of the paper. JL made critical contributions to the conception of the work, and extensive revisions of the paper. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by Samsung Advanced Institute of Technology, NRF grants funded by MSIT of Korea (Nos. 2017R1D1A1B03033591 and 2020R1A2C2015066), IITP grants funded by MSIT of Korea [No. 1711080972, Neuromorphic Computing Software Platform for Artificial Intelligence Systems, No. 2020-0-01336, Artificial Intelligence Graduate School Program (UNIST)], and Free Innovative Research Fund of UNIST (1.170067.01).

ACKNOWLEDGMENTS

This paper was an extension of work originally presented at DAC 2018 (Sim et al., 2018), with new contributions relevant to neuromorphic computing including a much more scalable and memory-integrating architecture and fault tolerance analysis.

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
- Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., et al. (2014). “Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '14* (New York, NY: ACM), 269–284. doi: 10.1145/2541940.2541967
- Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V. (2017). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circuits* 52, 127–138. doi: 10.1109/JSSC.2016.2616357
- Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., et al. (2014). cudnn: Efficient primitives for deep learning. *arXiv* 1410.0759.
- Courbariaux, M., and Bengio, Y. (2016). Binarynet: training deep neural networks with weights and activations constrained to +1 or -1. *CoRR* abs/1602.02830.
- Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Chodary, 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
- 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 *ISCA'16* (Seoul: IEEE Press), 243–254. doi: 10.1145/3007787.3001163
- Hawkins, J., and George, D. (2006). *Hierarchical Temporal Memory: Concepts, Theory and Terminology*. Technical report, Numenta.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Netw.* 2, 359–366. doi: 10.1016/0893-6080(89)90020-8
- 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>
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: convolutional architecture for fast feature embedding. *arXiv* 1408.5093. doi: 10.1145/2647868.2654889
- Judd, P., Albericio, J., Hetherington, T., Aamodt, T., Jerger, N. E., Urtasun, R., et al. (2015). Reduced-precision strategies for bounded memory in deep neural nets. *arXiv* 1511.05236.
- Judd, P., Albericio, J., Hetherington, T., Aamodt, T. M., and Moshovos, A. (2016). “Stripes: bit-serial deep neural network computing,” in *MICRO'16* (Taipei: IEEE), 1–12. doi: 10.1109/MICRO.2016.7783722
- Kim, K., Kim, J., Yu, J., Seo, J., Lee, J., and Choi, K. (2016). “Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks,” in *DAC'16* (Austin, TX), 124:1–124:6. doi: 10.1145/2897937.2898011

- Krizhevsky, A., and Hinton, G. (2009). *Learning Multiple Layers of Features From Tiny Images* (Toronto, ON: University of Toronto).
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *NIPS'12* (Tahoe City, CA), 1097–1105.
- LeCun, Y., Cortes, C., and Burges, C. (2010). *MNIST Handwritten Digit Database* (New York, NY: New York University).
- 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, J., Ren, A., Li, Z., Ding, C., Yuan, B., Qiu, Q., et al. (2017a). "Towards acceleration of deep convolutional neural networks using stochastic computing," in *ASP-DAC'17* (Chiba), 115–120. doi: 10.1109/ASPDAC.2017.7858306
- Li, Z., Ren, A., Li, J., Qiu, Q., Wang, Y., and Yuan, B. (2016). "DSCNN: hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *ICCD'16* (Scottsdale, AZ), 678–681. doi: 10.1109/ICCD.2016.7753357
- Li, Z., Ren, A., Li, J., Qiu, Q., Yuan, B., Draper, J., et al. (2017b). "Structural design optimization for deep convolutional neural networks using stochastic computing," in *DATE'17* (Lausanne), 250–253. doi: 10.23919/DATE.2017.7926991
- Lin, D., Talathi, S., and Annapureddy, S. (2016). "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning* (New York, NY), 2849–2858.
- 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
- Markram, H. (2012). The human brain project. *Sci. Am.* 306, 50–55. doi: 10.1038/scientificamerican0612-50
- McCulloch, W. S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* 5, 115–133. doi: 10.1007/BF02478259
- Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356
- Qian, W., Li, X., Riedel, M. D., Bazargan, K., and Lilja, D. J. (2011). An architecture for fault-tolerant computation with stochastic logic. *IEEE Trans. Comput.* 60, 93–105. doi: 10.1109/TC.2010.202
- Rahman, A., Lee, J., and Choi, K. (2016). "Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (Dresden), 1393–1398. doi: 10.3850/9783981537079_0833
- Ren, A., Li, Z., Ding, C., Qiu, Q., Wang, Y., Li, J., et al. (2017). "SC-DCNN: highly-scalable deep convolutional neural network using stochastic computing," in *ASPLOS'17* (Xi'an), 405–418. doi: 10.1145/3037697.3037746
- Ren, A., Li, Z., Wang, Y., Qiu, Q., and Yuan, B. (2016). "Designing reconfigurable large-scale deep learning systems using stochastic computing," in *ICRC'16* (San Diego, CA), 1–7. doi: 10.1109/ICRC.2016.7738685
- 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
- Sim, H., Kenzhegulov, S., and Lee, J. (2018). "DPS: dynamic precision scaling for stochastic computing-based deep neural networks," in *Proceedings of the 55th Annual Design Automation Conference, DAC 18* (New York, NY: Association for Computing Machinery). doi: 10.1145/3195970.3196028
- Sim, H., and Lee, J. (2017). "A new stochastic computing multiplier with application to deep convolutional neural networks," in *DAC'17* (Austin), 29:1–29:6. doi: 10.1145/3061639.3062290
- Sim, H., and Lee, J. (2019). Cost-effective stochastic mac circuits for deep neural networks. *Neural Netw.* 117, 152–162. doi: 10.1016/j.neunet.2019.04.017
- Sim, H., Nguyen, D., Lee, J., and Choi, K. (2017). "Scalable stochastic-computing accelerator for convolutional neural networks," in *ASP-DAC'17* (Chiba), 696–701. doi: 10.1109/ASPDAC.2017.7858405
- Stöckl, C., and Maass, W. (2020). *Classifying Images With Few Spikes per Neuron* (Ithaca, NY: Cornell University).
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA). doi: 10.1109/CVPR.2015.7298594
- Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2019). *A Tandem Learning Rule for Efficient and Rapid Inference on Deep Spiking Neural Networks* (Ithaca, NY).
- Yu, J., Kim, K., Lee, J., and Choi, K. (2017). "Accurate and efficient stochastic computing hardware for convolutional neural networks," in *2017 IEEE International Conference on Computer Design (ICCD)* (Boston, MA), 105–112. doi: 10.1109/ICCD.2017.24
- Zhakatayev, A., Lee, S., Sim, H., and Lee, J. (2018). "Sign-magnitude SC: getting 10× accuracy for free in stochastic computing for deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)* (San Francisco, CA), 1–6. doi: 10.1145/3195970.3196113
- Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., and Zou, Y. (2016). Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR* abs/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.

Copyright © 2020 Sim and Lee. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.



End-to-End Implementation of Various Hybrid Neural Networks on a Cross-Paradigm Neuromorphic Chip

Guanrui Wang, Songchen Ma, Yujie Wu, Jing Pei, Rong Zhao and Luping Shi*

Department of Precision Instrument, Center for Brain-Inspired Computing Research (CBICR), Beijing Innovation Center for Future Chip, Optical Memory National Engineering Research Center, Tsinghua University, Beijing, China

OPEN ACCESS

Edited by:

Huajin Tang,
Zhejiang University, China

Reviewed by:

Anup Das,
Drexel University, United States
Zhaofei Yu,
Peking University, China

*Correspondence:

Luping Shi
lpshi@tsinghua.edu.cn

Specialty section:

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

Received: 08 October 2020

Accepted: 08 January 2021

Published: 02 February 2021

Citation:

Wang G, Ma S, Wu Y, Pei J,
Zhao R and Shi L (2021) End-to-End
Implementation of Various Hybrid
Neural Networks on
a Cross-Paradigm Neuromorphic
Chip. *Front. Neurosci.* 15:615279.
doi: 10.3389/fnins.2021.615279

Integration of computer-science oriented artificial neural networks (ANNs) and neuroscience oriented spiking neural networks (SNNs) has emerged as a highly promising direction to achieve further breakthroughs in artificial intelligence through complementary advantages. This integration needs to support individual modeling of ANNs and SNNs as well as their hybrid modeling, which not only simultaneously calculates single-paradigm networks but also converts their different information representations. It remains challenging to realize effective calculation and signal conversion on the existing dedicated hardware platforms. To solve this problem, we propose an end-to-end mapping framework for implementing various hybrid neural networks on many-core neuromorphic architectures based on the cross-paradigm Tianjic chip. We construct hardware configuration schemes for four typical signal conversions and establish a global timing adjustment mechanism among different heterogeneous modules. Experimental results show that our framework can implement these hybrid models with low execution latency and low power consumption with nearly no accuracy degradation. This work provides a new approach of developing hybrid neural network models for brain-inspired computing chips and further tapping the potential of these models.

Keywords: hybrid neural networks, cross-paradigm computing, neuromorphic chip, mapping framework, end-to-end implementation

INTRODUCTION

Neural networks have been widely used to deal with intelligence problems. In general, they can be divided into non-spiking artificial neural networks (ANNs) (Lecun et al., 2015) and spiking neural networks (SNNs) (Maass, 1997; Ghosh-Dastidar and Adeli, 2009). These two types of neural models are distinct in information representation and processing. In ANNs, information is propagated with multi-valued data. Intensive representation makes ANNs achieve high accuracies in a myriad of tasks, such as image classification (He et al., 2016), speech recognition (Lam et al., 2019), and action recognition (Wu et al., 2016). In contrast, SNNs encode information in event-driven binary spike trains. Through internal neuron dynamics to memorize spatio-temporal information, SNNs show advantages in various scenarios with rich temporal information and sparse data streams (Shi et al., 2017; Haessig et al., 2018; Wu et al., 2019). Owing to their different advantages, in recent years there is a growing trend of integrating ANNs and SNNs to explore hybrid neural networks

(HNNs) toward artificial general intelligence (Marblestone et al., 2016; Zhang et al., 2016; Ullman, 2019). For example, in some cases of event-driven tasks (Srinivasan and Roy, 2019; Lee et al., 2020), researchers use SNN modules for abstracting sparse temporal information, and further combine ANN modules for improving the classification performance. Similarly, in some cases of static image processing tasks (Kheradpisheh et al., 2018; Chancán et al., 2020), researchers use ANN modules to extract the edge contrasts in images and further process them with SNN modules for low power consumption. Besides, ANNs and SNNs also work collaboratively to perform complex tasks in Pei et al. (2019); Yang et al. (2019).

Hybrid neural networks have a promising perspective on the development of artificial general intelligence. However, by far these models are mainly studied and implemented on general-purpose platforms (i.e., CPU or GPU) (Kheradpisheh et al., 2018; Srinivasan and Roy, 2019; Chancán et al., 2020; Lee et al., 2020). On the other side, HNNs retain the basic properties of neural networks, being promising in high-efficiency implementation on domain-specific hardware platforms (Sze et al., 2017). However, their unique cross-paradigm mechanisms, such as the mixed dataflow of multi-valued data and spike trains, hinder the implementation on dedicated platforms, thereby slowing down the exploration of diverse cross-paradigm integration. Thus, it is highly expected to develop a general scheme of implementing HNNs on dedicated platforms for high efficiency, which can facilitate the iteration of software and hardware co-optimization and eventually promote the development of hybrid neural models.

There are two challenges in implementing HNNs on dedicated hardware platforms. The first is to support the simultaneous execution of ANN and SNN computing paradigms. In the traditional ANN or SNN field, each has its respective hardware platforms to support their efficient execution, e.g., deep learning accelerators for ANNs (Chen et al., 2014; Han et al., 2016; Jouppi et al., 2017; Chen et al., 2019) and neuromorphic chips for SNNs (Furber et al., 2014; Merolla et al., 2014; Davies et al., 2018). However, due to the significant differences between ANNs and SNNs in terms of information representation, computation philosophy and memory organization, the basic operators and data transmission methods of these two types of platforms are incompatible. Therefore, neither of the above hardware platforms can simultaneously support the execution of ANNs and SNNs, which impedes the implementation of HNNs. The second is the hybrid data interactions between ANNs and SNNs. In HNNs, the hybrid data interaction modules connect ANNs and SNNs, which have a non-negligible impact on the performance of the models when implemented on a hardware platform. Usually, the hybrid data interaction results in at least two-fold computational costs: (1) realizing signal conversion between multi-valued data and spike trains will bring extra resource consumption and execution delay; (2) the signal conversion and timing configuration will in turn affect the resource consumption and execution time of ANN and SNN modules. In current dedicated hardware platforms, signal conversion at the input interface needs to be implemented when the external data cannot match the information format transmitted and processed internally. Therefore, extra devices

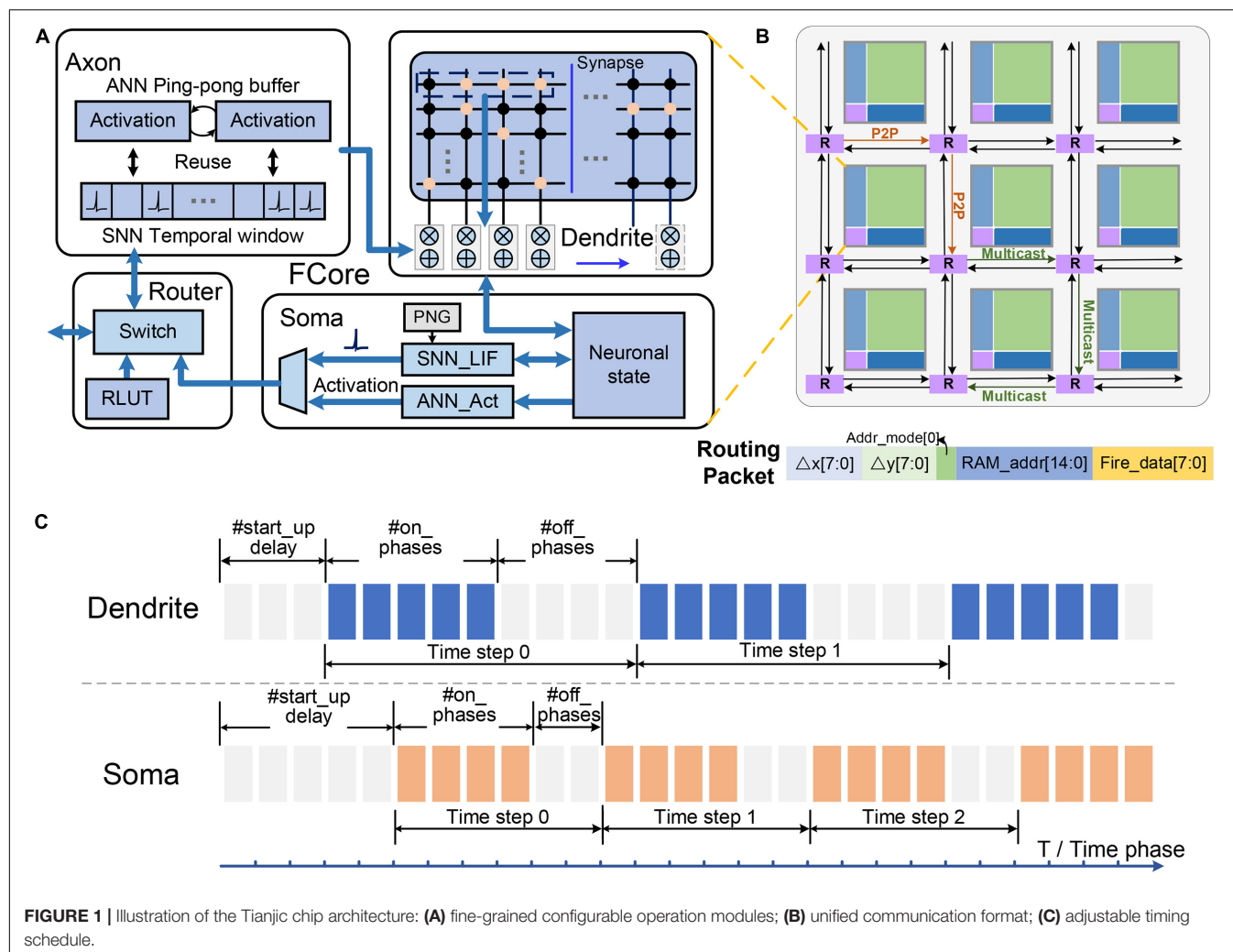
and resources are usually required, such as “spike generator” (Esser et al., 2016; Shukla et al., 2019) or “frame maker” (Shukla et al., 2019). However, this separative method will not only destroy the continuity of hardware execution to a certain extent, but also make it difficult to comprehensively measure and evaluate the implementation cost of signal conversion and network computing via a unified standard.

In this paper, we provide a systematic scheme of implementing HNNs on many-core neuromorphic architectures based on software-hardware cooperation from the perspectives of hardware features and mapping framework. First, we use a new type of cross-paradigm Tianjic chip (Pei et al., 2019) as the hardware infrastructure. From the aspects of basic operations, communication method, and timing execution mechanism, the hardware features that support the implementation of HNNs are abstracted. On this basis, we propose an end-to-end mapping framework to implement HNNs on many-core neuromorphic chips. Inspired by the modular approach, we divide the implementation of HNNs into the pure computing modules of single ANNs and SNNs, and the signal conversion modules between them. The pure computing modules can be realized by using the existing single-paradigm mapping methods (Esser et al., 2013; Deng et al., 2018; Ji et al., 2018). To realize the hybrid data interactions between ANNs and SNNs, configuration schemes for four typical signal conversions methods are established. Besides, we also develop a global timing adjustment mechanism to match the different working periods among these modules. Taking some HNN models as examples, we analyze their performance in terms of resource overhead, running speed, and energy consumption when deployed on the Tianjic chip. This implementation framework provides a generic approach for developing hybrid neural models through the hardware-software collaboration.

The rest of this paper is organized as follows. Section “Hardware Infrastructure” introduces the basic operation, communication format, and timing schedule mechanism of the Tianjic chip from the perspective of hardware feature abstraction. Section “End-to-End Mapping Framework” shows the characteristics of neural networks’ execution on many-core neuromorphic chips, and presents the proposed end-to-end mapping framework for hybrid neural models. The resource overhead, timing analysis, and energy consumption of the example hybrid networks are reported in Section “Experimental Results.” Finally, we come up with the overall conclusions and carry out further discussions in Section “Conclusion and Discussion.”

HARDWARE INFRASTRUCTURE

Tianjic adopts a unified, configurable, and scalable architecture to support cross-paradigm computing, which provides a general platform for the separative execution or hybrid computing of ANNs and SNNs. In this section, we will briefly introduce the overall architecture of the Tianjic chip (see **Figure 1**), including its basic operation, communication format, and timing schedule mechanism which support our mapping framework.



Fine-Grained Configurable Operation Modules

Functional core (FCore) is the basic unit of the Tianjic chip, which consists of four modules, including an axon for input organization, a dendrite (with synapses) for integration operations, a soma for non-linear neuronal transformation, and a router for activation transmission (Figure 1A). Each module can be configured to work in different modes or perform different operations, which enables the chip to support both ANN and SNN models. Among these modules, the dendrite and the soma are the main computing engines. Equipped with the synapse memory, the dendrite constitutes a 256×256 virtual crossbar, which can realize various vector and matrix operations. Table 1 lists the vector and matrix operations used in this paper, including vector-matrix multiplication (VMM), vector-vector accumulation (VVA) and vector buffering (VB).

The calculation results of the dendrite are updated into a memory shared with the soma and the soma generates the neuronal output according to the updated membrane potential. By combining some basic calculations, the soma can realize a variety of non-linear transformations in different modes of

ANNs and SNNs. In an ANN-mode soma, arbitrary activation function can be supported by a configurable lookup table (LUT). In an SNN-mode soma, its internal operation corresponds to the leaky-integration-and-fire (LIF) operation of spiking neurons. Furthermore, some more complicated operations, such as threshold adaption and random firing, can also be enabled

TABLE 1 | Integration and transformation operations in Tianjic.

	Mode	Definition
Dendrite operation	VMM	$y = W \cdot x$
	VVA	$y = \sum_i x_1 + x_2 + \dots + x_n$
	VB	$y = x$
Soma transformation	LUT_fun	$y = f(u + b)$
	LIF_fun	Leaky-integration-and-fire operation
Membrane potential	change	Update membrane potential after soma transformation
	keep	Membrane potential keeps unchanged after soma transformation

through specific configurations. After the transformation of the soma, the update mode of the shared memory can be divided into *keep* or *change*, corresponding to whether the membrane potential stored therein will be changed according to the calculation result of the soma.

Unified Communication Format

These FCores are connected by a multifunctional and scalable routing network, and arranged in a 2D mesh topology. The routing network is composed of the router and the axon in each FCore, which are responsible for sending and receiving information respectively. In the router, both artificial and spiking neurons transmit their information through a unified communication format. In addition to the address and control information in the traditional AER (Address-Event Representation) protocol (Mahowald, 1993), the routing packet can also carry different multi-valued data in this unified communication format. Specifically, this multi-valued data represents the efferent activation value of artificial neurons in the ANN mode and the state information of spiking neurons (e.g., membrane potential) in SNN mode. Notably, the router generates a packet only when a spike is generated and needs to be transmitted, which is in an even-driven manner.

As shown in **Figure 1B**, these data packets can be delivered to single or multiple arbitrary target FCores through point-to-point (P2P) or multicast routing. When arriving at the destination FCore, the packet is decoded to a binary spike or multi-valued activation according to the working mode of the local axon. In the ANN mode, the axon directly obtains multi-value activations from routing packets and store them in a ping-pong buffer. In the SNN mode, the axon stores spike trains of each input within a historical temporal window. Via a timing factor calculator (TFC), the spike trains can be weighted and summed according to the timing factors. Based on this cross-paradigm and unified data communication format, we can easily realize the basic connection structure that supports mixed dataflows as described in section “Execution of Neural Networks on Neuromorphic Chips.”

Adjustable Timing Schedule

The timing execution mechanism in the Tianjic chip has two typical characteristics: the reconfigurable phase pattern in an execution time step and the independent working schedule of each FCore’s modules.

Reconfigurable phase pattern in an execution time step:

There are two levels of execution period in Tianjic, which are time phase and time step. The time phase is a basic computational period to perform a round of computation, and the time step includes multiple time phases and therefore is a higher level of execution period. As shown in **Figure 1C**, the number of time phases in a time step and their on-off (enable and disable) pattern are controlled by timing registers (i.e., start-up delay, #on_phases and #off_phases). This configurable phase pattern provides a flexible support for matching different execution periods of ANNs and SNNs.

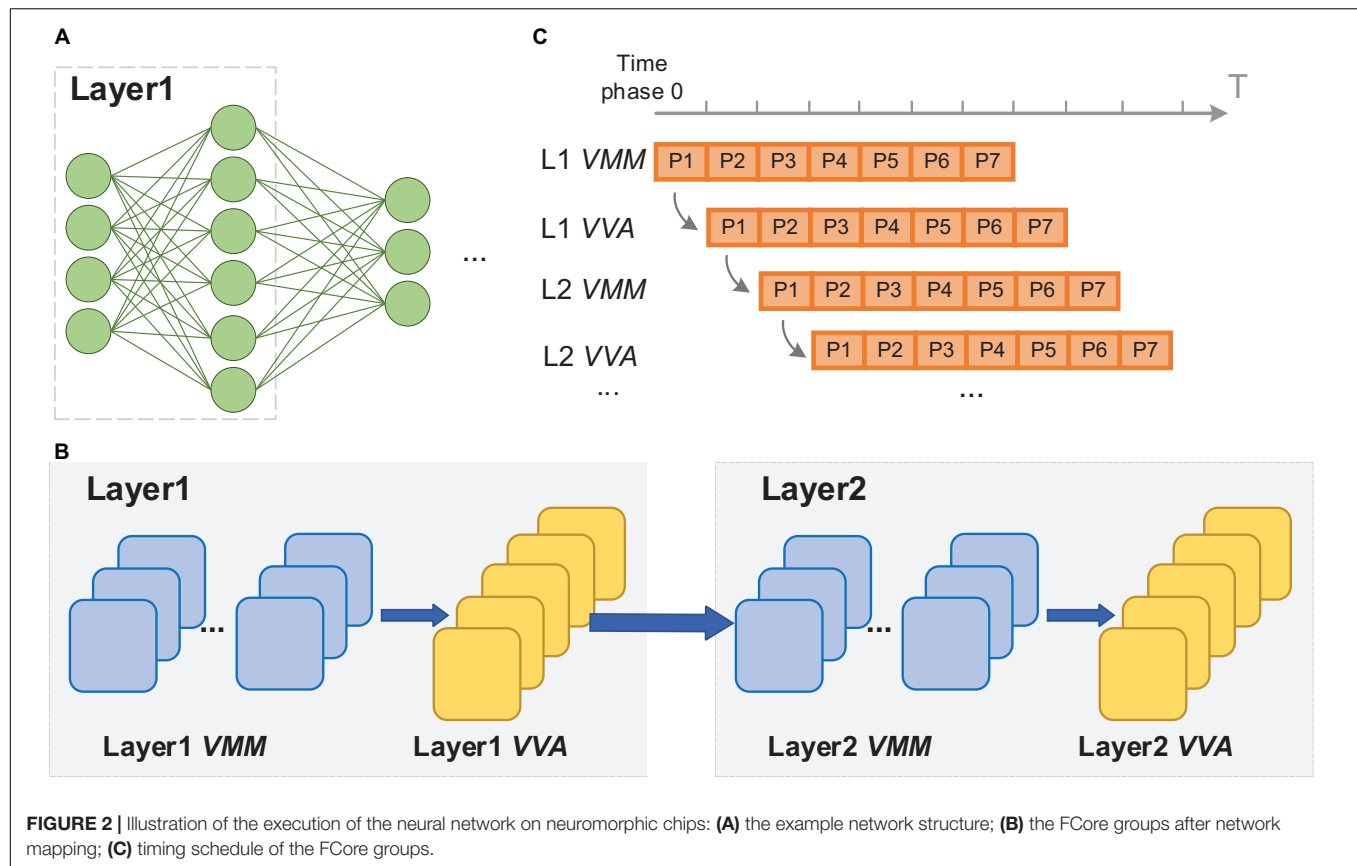
Independent working schedule of each FCore’s modules: In a time phase, the dendrite integrates the inputs stored in the axon and updates the membrane potential into the shared memory, meanwhile the soma performs non-linear transformation given the integrated membrane potential. In each FCore, the phase patterns of the dendrite and the soma can be configured independently. In this way, different timings of input and output processing can be implemented in the same FCore to perform signal conversion between spike trains and multi-valued activations.

END-TO-END MAPPING FRAMEWORK

Before introducing the mapping scheme, we briefly recall the execution mechanism of single-paradigm neural network models on many-core neuromorphic chips. Then, we introduce the main design features of our end-to-end mapping framework, which enable a high-performance mapping of HNNs on many-core neuromorphic chips. With configurable FCores, three basic connections are designed to support the mixed dataflows. By using the divide-and-conquer strategy, we further divide the implementation of HNNs into the pure computing modules of single ANNs and SNNs, and the hybrid data interaction modules between them. The pure computation can be implemented using the existing mapping methods for the single paradigm. To solve the problem of the hybrid data interaction, we construct the configuration schemes for typical signal conversion methods and a global timing adjustment mechanism between these different modules.

Execution of Neural Networks on Neuromorphic Chips

Generally, the implementation of neural networks on many-core neuromorphic chips is achieved by utilizing spatial mapping methods, in which the calculations in different layers are realized via the allocated FCore groups. These FCore groups continuously process the input data in a pipelined manner. Taking a fully connected network (**Figure 2A**) for illustration, we present this process in **Figure 2**. As shown in **Figure 2B**, in each layer, the calculations between input activations and weights are split into multiple spatial VMM operations due to the limited fan-in capability (the number of inputs a neuron can handle) and fan-out capability (the number of outputs a neuron can drive) of each FCore. Therefore, each VMM FCore obtains partial calculation results, and extra VVA FCores are required to accumulate the corresponding neurons’ partial states. In this layer-wise splitting manner, the workload of the original network will be mapped to a combination of FCore groups that perform different operations. **Figure 2C** exhibits the execution timing of these FCore groups, wherein each FCore performs the same operation repeatedly and continuously at every time phase. In addition, the data is continuously propagated and processed among FCore groups along the depth dimension of the network. When the calculation results are sent to the next FCore group for processing, the current FCore group can start the processing the following input sample at the same time,



achieving an efficient pipelined processing. This inter-group pipeline brings high throughput, and is decoupled with the network depth. Most single-paradigms of ANNs and SNNs follow this method when implemented on many-core neuromorphic chips (Akopyan et al., 2015; Ji et al., 2018; Shao et al., 2019; Jiao et al., 2020). It's worth noting that since SNNs use the binary spike for information representation, the multi-valued data preferred by ANNs are encoded into spike train with a time window. When an SNN is mapped to many-core neuromorphic chips, each FCore group needs to perform repeatedly along the T_w (length of time window) phases to process a frame image or feature map.

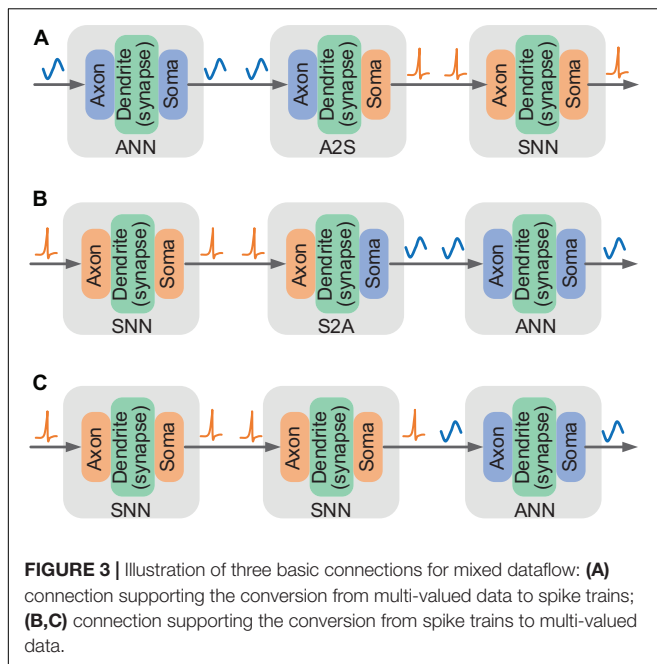
Overall, to end-to-end implement hybrid networks on neuromorphic chips, it requires to not only establish connections among FCore groups that support mixed dataflows, but also coordinate the different requirements of execution phases between ANNs and SNNs.

Basic Connections for Mixed Dataflow

In order to support mixed dataflows of multi-valued data and spike trains, we design three basic connections based on the fine-grained configuration of input and output modes of the FCores. According to the configuration, four kinds of FCores with different output and input relations can be formed. When the axon and the soma are configured in the same mode (either ANN or SNN mode), the FCore processes pure ANN signals in multi-value data or SNN signals in binary spikes respectively,

which can be allocated to perform the calculation in ANN and SNN modules of HNNs. We call such FCores working in the pure-ANN or pure-SNN mode. When data conversion is needed, the axon and the soma are configured to work in different modes, forming hybrid FCores with ANN-input and SNN-output (A2S) or with SNN-input and ANN-output (S2A). These hybrid FCores can be used to implement the conversion between multi-valued data and spike trains, thus supporting hybrid modeling and interaction in HNNs.

By virtue of the unified routing infrastructure, these different types of FCores can formulate a variety of basic connections that enable to process single and mixed dataflows. When the soma of the pre-connected FCore and the axon of the post-connected FCore are configured in the same working mode, data can be directly transmitted between them. **Figures 3A,B** depict the connections that can be used to realize conversion from multi-valued data to spike trains and vice versa, respectively. In **Figure 3A**, the first and last FCores work in the pure-ANN and pure-SNN modes respectively. The intermediate FCore, working in the A2S mode, receives multi-valued data and converts it into binary spikes via designed internal operations. Similarly, the intermediate FCore working in the S2A mode converts the spike trains to multi-valued data in **Figure 3B**. In addition, as shown in **Figure 3C**, an ANN axon can also directly connect with an SNN soma and access neuronal state information from the routing packet. In this connection, the signal conversion occurs during data transmission and reception.



By constructing these basic connections for mixed dataflows, the signal conversion can be naturally performed on the critical data path of FCores without extra devices. In this manner, the signal conversion of the model can be realized on the same carrier as other components. Its effects on the execution performance of the model can be directly displayed.

Configuration Schemes for Signal Conversion

In general, different signal conversion methods are applied according to the algorithm details. The data interaction between ANNs and SNNs requires signal conversion between spike trains and multi-valued data. The commonly used data conversion methods can be summarized as: probabilistic sampling, ANN-SNN encoding layer, space expansion, and time accumulation (Deng et al., 2020b). Probabilistic sampling and ANN-SNN encoding layer are responsible for the conversion from multi-valued data to spike trains, while spatial expansion and temporal accumulation are responsible for the conversion from spike trains to multi-valued data. We establish configuration schemes for these typical signal conversions (as illustrated in **Figure 4**), whose operations are mapped into the configurations of the working modes of the building blocks in each FCore.

Probabilistic sampling converts multi-valued data to spike trains through an element-by-element operation. At each time phase in the time window, a random vector with the same size as the original multi-valued data is generated. After comparison, the multi-valued data is sampled as binary spikes. The FCores used for realizing probabilistic sampling work in the A2S mode. The axons' input data is directly transmitted to the soma through VB operation. Their somas work in the SNN mode and the random threshold is enabled. Random numbers with uniform distribution are generated as the threshold of membrane potential, so as to

realize the sampling of input multi-valued data. The update mode of membrane potential is set to *keep*, so that the multi-valued data can be saved after the first reception, which will be converted into a spike train latter via soma sampling with multiple time phases.

ANN-SNN encoding layer can be regarded as a special SNN layer that can process multi-valued input data. Different from probabilistic sampling, the encoding layer adopts a rank-order coding format. Before being converted into spikes, the original multi-valued data is processed by a global calculation in advance, which can be a fully connected calculation (Bellec et al., 2018), a convolution (Wu et al., 2019), or a difference-of-Gaussians (DoG) (Kheradpisheh et al., 2018). The integration results are continuously accumulated onto the membrane potential of the output neuron, and the neuron fires a spike once the membrane potential exceeds the firing threshold at any time phase. In order to reduce redundant integration operations, we implement the encoding layer in two FCore groups: integration FCores and conversion FCores. In integration FCores, the dendrite performs the VMM operation at the first time phase in a time window and stores the integration results in the shared memory. The ANN soma of FCores continuously sends out the integration results (or their partial sums) to the conversion FCores, where spikes are generated through the LIF operation in the SNN soma.

Spatial expansion directly transfers the spatio-temporal two-dimensional spike pattern into a static binary image. Each "1" in the static binary image corresponds to the existence of a spike in the original spike pattern. In hardware implementation, this spatial expansion method needs to buffer and rearrange data from different time steps. To solve this problem, we establish a self-feedback routing connection in the conversion FCore. The input spikes are shifted and sorted at each time phase. The rearranged spikes are sent to the downstream adjacent ANN axon through multicast routing as binary image data. However, in this conversion method, the scale of the newly generated binary data is proportional to the length of the time window. Consequently, it will consume massive computing and storage resources in the case of a large time window.

Temporal accumulation directly accumulates the spike train of each input node within the time window into a multi-valued data. This kind of conversion can be regarded as the reverse process of rate coding, and the converted multi-valued data has the same spatial size as the input spikes. This conversion can be realized by the TFC in axon. Spike accumulation for T_w length is enabled in TFC, and the corresponding time factors are configured as 1. In this way, the data transmitted to the dendrite module for integrated calculation is the multi-valued data obtained by accumulation. In the conversion FCore, only the axon is used for signal conversion, and the dendrite and the soma can directly execute the subsequent ANN calculations.

The configuration schemes for these typical signal conversions are summarized in **Table 2**. In the hybrid networks, the signal conversion operations are mapped into the configuration of the FCore according to the algorithmic computing operations. The model can generate the corresponding special conversion FCore groups in accordance with different requirements to form a connection with hybrid dataflows.

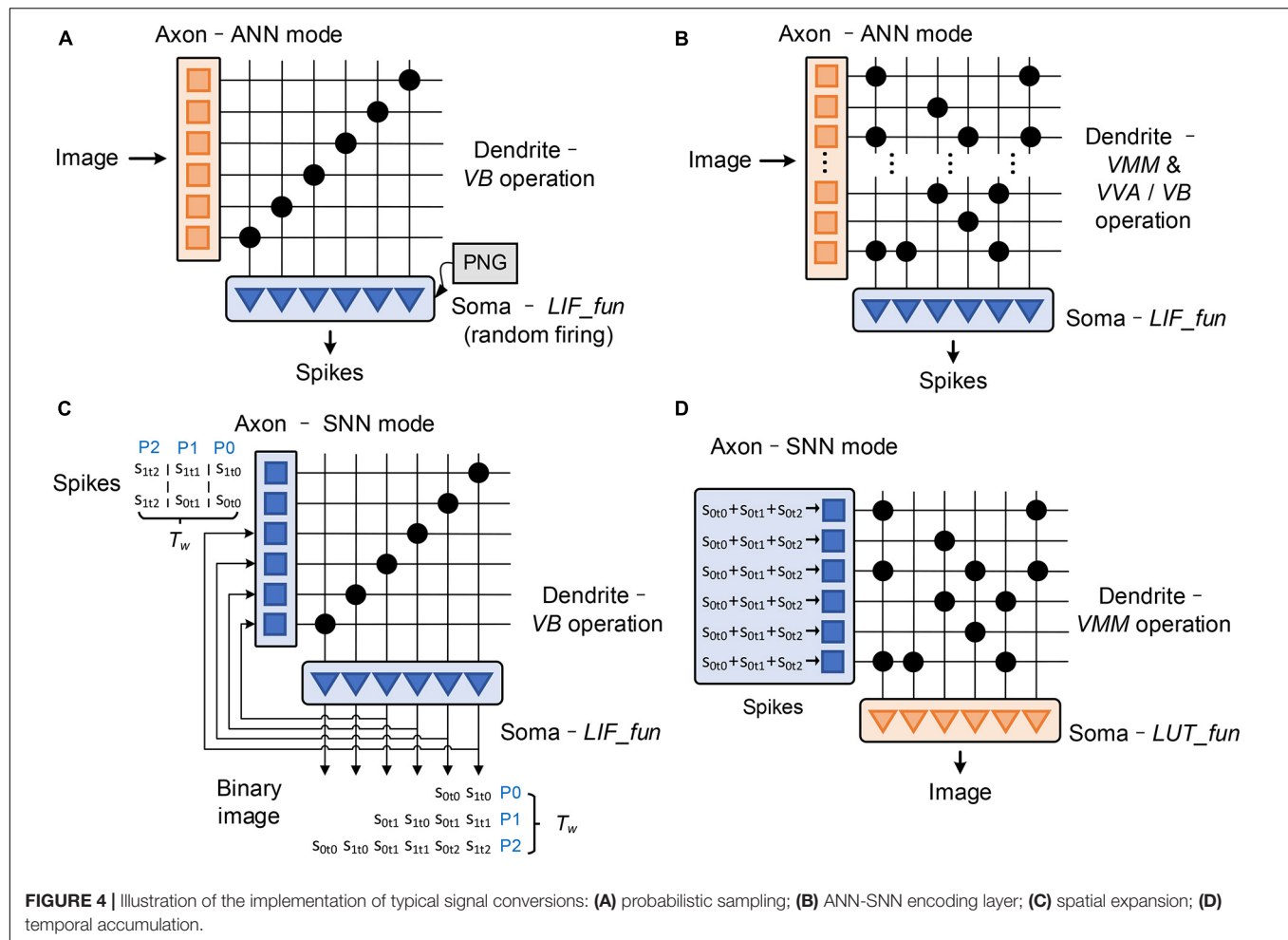


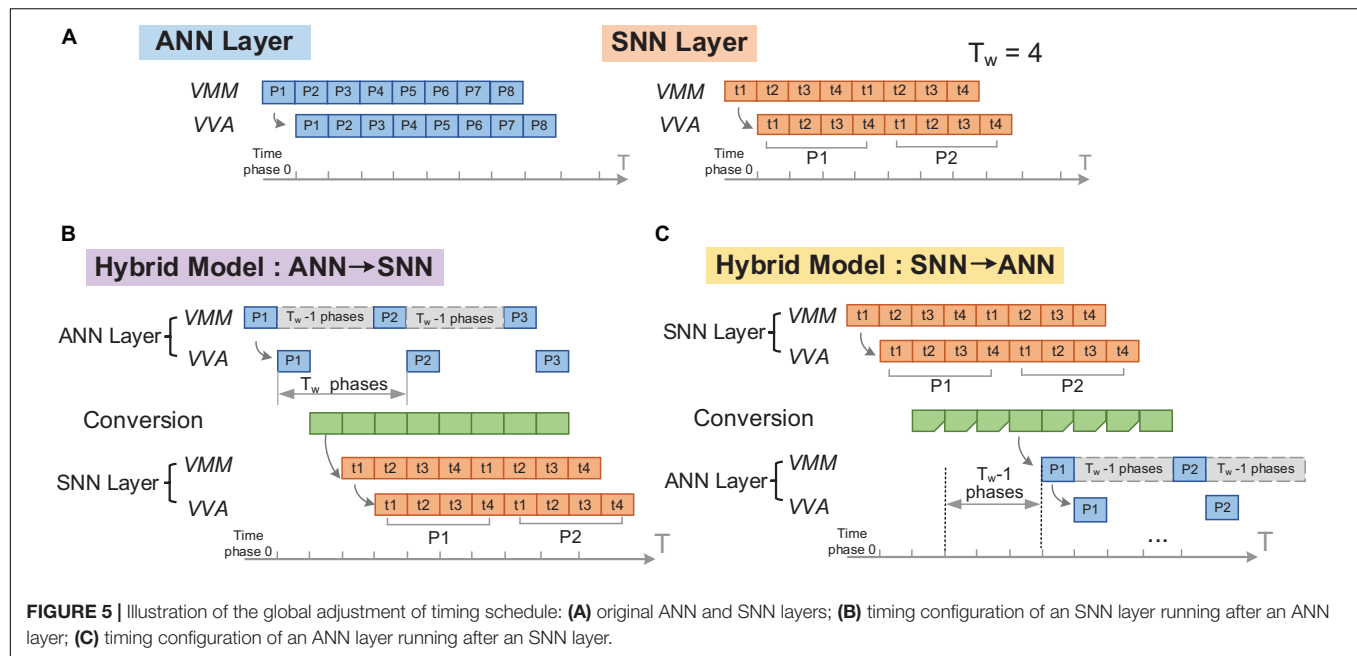
TABLE 2 | Summary of the configuration schemes for typical signal conversions.

		Probabilistic sampling	Encoding Layer		Spatial expansion	Temporal accumulation
			Integration	Conversion		
Axon	Mode	ANN	ANN	ANN	SNN	SNN (with TFC)
Dendrite	Operation	VB	VMM	VVA/VB	VB	VMM
	Enable time	the first phase	the first phase	always on	always on	always on
Soma	Mode	LIF_fun (random firing)	LUT_fun	LIF_fun	LIF_fun	LUT_fun
	Enable time	always on	always on	always on	the last phase	the last phase
Membrane potential		keep	keep	change	change	change

Global Adjustment of Timing Schedule

As introduced in Section “Execution of Neural Networks on Neuromorphic Chips,” ANNs and SNNs have different pipelining cycles (1 phase in ANN and T_w phases in SNN) and need to adjust the timing schedule when combining the ANN layers and SNN layers together. In hybrid models, when an ANN layer runs ahead of an SNN layer, the ANN layer is expected to wait for the SNN layer to execute T_w times continuously before transmitting the next data. Similarly, when an ANN

layer runs behind an SNN layer, the ANN layer only needs to perform the calculation once after the SNN layer continuously executes T_w times. Therefore, it is preferred that in hybrid models, the ANN only starts at a suitable time for effective calculation and data transmission, instead of performing the same operation phase by phase repeatedly. Hence, we use the configurable phase pattern introduced in Section “Adjustable Timing Schedule” to realize a global timing adjustment of the timing schedule of the FCores.



We demonstrate this global adjustment mechanism in **Figure 5**. **Figures 5B,C** illustrate the situation of “ANN layer to SNN layer” and “SNN layer to ANN layer” respectively. In both situations, the time step is set to include T_w time phases. In each time step, the FCore groups corresponding to the SNN calculation execute continuously, while the FCore groups corresponding to the ANN calculation only start at the first phase. Therefore, the phase patterns of these two types of FCore groups will be configured as $\#on_phases = T_w$, $\#off_Phases = 0$ and $\#on_phases = 1$, $\#off_phases = T_w-1$ respectively. Due to the delay of data transmission, different FCore groups will have different start-up time. Generally, this start-up delay is the number of FCore groups that need to pass before data arrives. However, in the mapped structure, as long as there exists an FCore group for ANN-SNN conversion, the subsequent start-up delay needs to increase the extra time required by the ANN to wait for SNN to process data (i.e., T_w-1 time phases, see **Figure 5C**). After setting the correct start-up delay, the ANN and SNN in the hybrid model can continually process each frame of input data in a pipelined manner, “step” by “step.”

As shown in the **Table 2**, following the same method, the dendrite and the soma in the conversion FCores are also configured to have different phase patterns to deal with the intra-FCore mixed dataflow. These timing patterns are eventually transformed into the configuration of timing registers in each FCore. Such a global timing adjustment can not only keep the original pipeline mechanism, but also reduce redundant calculations.

EXPERIMENTAL RESULTS

Experimental Setup

We have built some examples of hybrid networks to verify our mapping framework and illustrate the implementation results.

TABLE 3 | Network models used in the experiments.

Dataset	Structure	Conversion Method	Name
MNIST (28×28)	MLP	Probabilistic Sampling	Model 1
		Encoding Layer	Model 2
	LeNet	Probabilistic Sampling	Model 3
		Encoding Layer	Model 4
NMNIST ($34 \times 34 \times 2$)	MLP	Spatial extension	Model 5
		Temporal accumulation	Model 6
	LeNet	Spatial extension	Model 7
		Temporal accumulation	Model 8

We chose MNIST (LeCun et al., 1998) and NMNIST (Orchard et al., 2015) data sets to demonstrate the proposed conversion approach. Each digit sample in MNIST is a 28×28 grayscale image and NMNIST is a neuromorphic version of MNIST with a spike pattern size of $34 \times 34 \times 2$ at each time step. As for the network structure, we chose the fully connected structure of input-512-512-10 and the convolutional neural network structure of LeNet (LeCun et al., 1998) (input-6c5-AP2-16c5-AP2-120-84-10). The overall settings of input data, network structure and signal conversion method are shown in the **Table 3**.

Figure 6 shows the settings of signal conversion in these models. From Model 1 to Model 4, the signal conversion from multi-value data to spike trains happens in the first layer of the network. At the output of these models, the spikes within the time window are accumulated and converted into multi-valued data to obtain the classification results. In Model 5 and Model 6, the signal conversion from spike trains to multi-valued data occurs in the connection between the first and second hidden layers. In Model 7 and Model 8, the signal conversion is performed between the last pooling layer and the fully connected classifier.

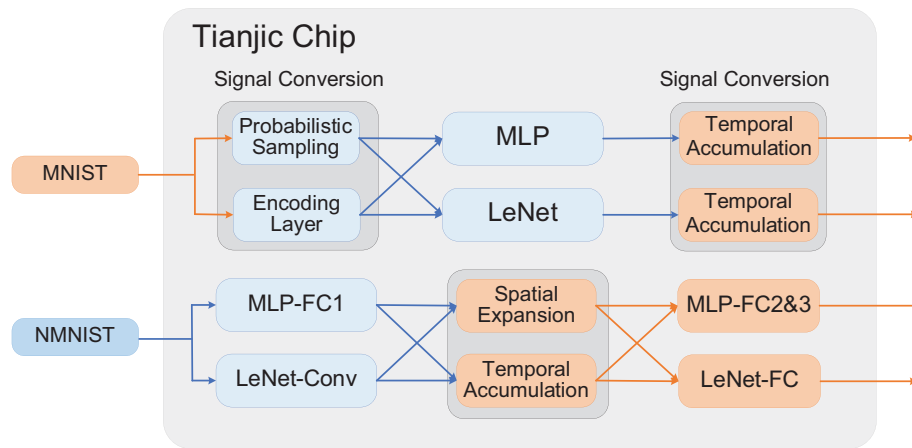


FIGURE 6 | Illustration of the configuration of the models.

We developed a mapping compiler in python to automatically implement the partition of networks, generation of conversion FCores, and global timing adjustment in our mapping framework. We used a direct quantization method for these hybrid models. We first loaded a pre-trained model with FP32 weights, and then followed the quantization method in Yang et al. (2020) to re-train the model by quantizing the weights into INT8 precision during each weight update. After this process, we can obtain the resource utilization report of network and the binary file for the chip configuration. At last, the configuration file is downloaded into the chip for execution, where the latency and power consumption of network execution can be measured accurately. We use a single-chip PCB equipped with an Altera Cyclone 4 FPGA as the test board. The input data is pre-stored in an SDRAM on the board, and injected into the chip through FPGA whiling testing. At 300 MHz clock, 16.8 μ s is needed for a time phase.

Analysis of Resource Consumption of Various Hybrid Models on Tianjic

After mapping, these hybrid network models are transformed into the connections between FCores with four different input and output types, as described in Section “Basic Connections for Mixed Dataflow.” **Figure 7** shows the resource utilization of different types of FCores in each hybrid model. When mapping the SNN part of these hybrid models, according to the method in Pei et al. (2019); Deng et al. (2020a), we transfer the partial sums calculated by VMM in the form of multi-valued data. This method can avoid the decrease of accuracy caused by the fan-in limitation of FCores in the mapping process. Therefore, when the number of input neurons in an SNN layer exceeds the fan-in of FCore, there exist S2A-type VMM FCores and A2S-type VVA FCores in the mapping result. Additionally, we define the ratio of effective computing FCores as the ratio of the number of FCores that perform network computing and the number of the total FCores. The larger the ratio of effective computing FCores

is, the smaller the extra cost of signal conversion in hybrid models consumes.

In a connection with the signal conversion from multi-valued data to spike trains, the resource consumption required by the probabilistic sampling and ANN-SNN encoding layer is determined by the number of input and output neurons in the connection, respectively. In the MLP structure, the conversion takes place in a 784-512 connection. Therefore, compared with Model 2, Model 1 uses more A2S-type FCores. Similarly, in the LeNet structure, the size of input and output in the connection with signal conversion is $28 \times 28 \times 1$ and $24 \times 24 \times 6$, respectively, which leads to Model 4 consuming more conversion FCores than Model 3. In the implementation of probabilistic sampling and ANN-SNN encoding layer, the signal conversions are carried out through additional FCores. From **Figure 7**, we can see that the ratio of effective computing FCores in Model 1~Model 4 are 84%, 96%, 92%, and 81%, respectively. This result also indicates that with the increase of the scale of converted signals, the proportion of the FCores that undertake the network computation in the hybrid model decreases.

In the implementation of spatial expansion, the conversion FCores work in the SNN mode to buffer and rearrange input spikes. As mentioned above, the FCores consumed by spatial expansion will increase with the increase of the SNN time window. To observe this trend, we show the resource consumption of Model 5 and Model 7 when T_w equals 2, 6, 10, and 14 respectively (shown as -T2, -T6, -T10 and T14 in **Figure 7**). As we can see, with the increase of T_w , the number of SNN-type FCores increases concomitantly, which is caused by the increase of the number of occupied conversion FCores. Furthermore, due to the growth of the scale of the converted binary image, the number of ANN-type FCores that are used to perform subsequent ANN calculations also increases. It is observed that, under the joint influence of these two kinds of growth, the ratio of effective computing FCores of the network finally shows an obvious downward trend. In the implementation of spatial expansion, only axons in the FCores are used, which does not affect the subsequent calculation of dendrites and somas.

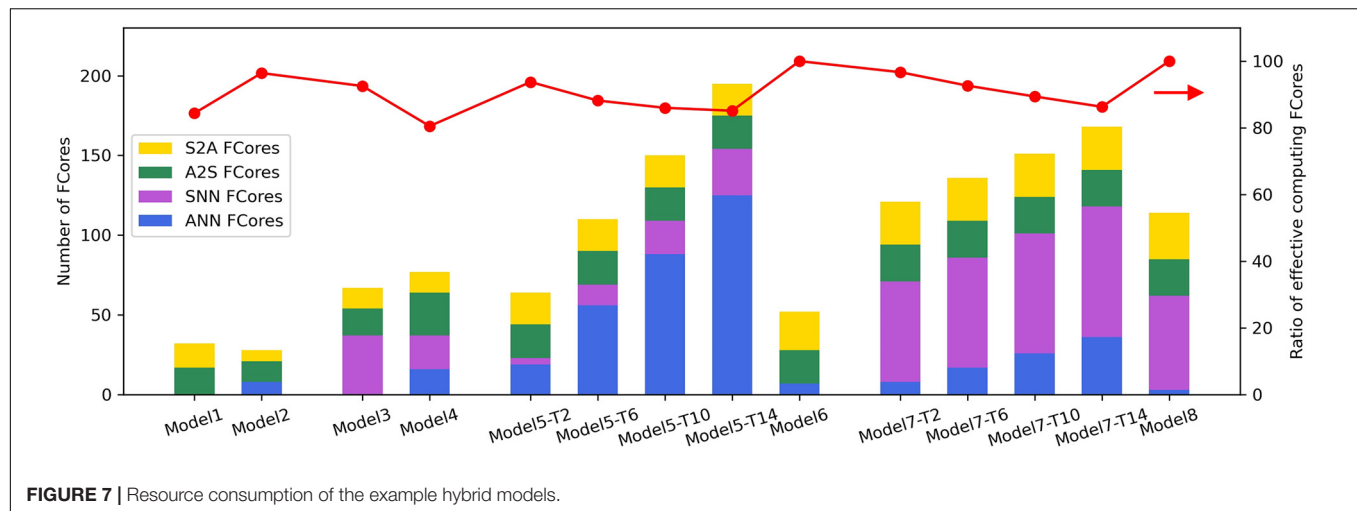


FIGURE 7 | Resource consumption of the example hybrid models.

As a result, the ratios of effective computing FCores are both 100% in Model 6 and Model 8. Regardless of the types of FCores, the resource consumption of these two models is the same as that of a single-paradigm network with the same structure.

Comparison of Performance on Tianjic and GPU

We tested the implementation of these hybrid models on GPU (Nvidia RTX 2080Ti) and the Tianjic chip, respectively, and summarized the outcomes in **Table 4**. The time window of SNNs in these hybrid models was set as 10. These hybrid models ran on GPU with the default FP32 precision. While running on Tianjic chip, all the weights and output activations were quantized to INT8. It is observed that there is no evident difference in recognition accuracy between the fixed-point network implemented on the Tianjic chip and the floating-point network running on GPU. In some models (i.e., Model 2 and Model 7), the accuracy on the Tianjic chip was slightly improved, which is owing to the regularization effect of quantization. Generally speaking, the accuracy of the model varies within 0.15%, which is almost negligible.

TABLE 4 | Execution performance of different implementations on Tianjic and GPU.

	GPU (Nvidia RTX 2080Ti)		Our Implementation	
	Acc. (%)	Latency (ms)	Acc. (%)	Latency (ms)
Model1	98.70	6.84	98.69	0.286
Model2	98.20	6.02	98.22	0.269
Model3	99.15	14.27	99.15	0.319
Model4	99.19	13.22	99.13	0.319
Model5	98.43	3.56	98.41	0.252
Model6	98.36	3.22	98.30	0.269
Model7	98.27	8.79	98.28	0.302
Model8	98.97	8.87	98.85	0.319

It is worth noting that, while running on Tianjic, all these models consume about 300 ms latency to process one frame of data. Compared with GPU, the processing speed is increased by an average of 20 times. This is mainly due to the high computational parallelism of the many-core architecture in the Tianjic chip. In all these models, the average power consumption is below 400 mW, verifying the advantage of low power consumption compared with GPU (usually with a dynamic power of 1~100 W).

Through the comprehensive comparison, we can conclude that when implemented on the Tianjic chip, the hybrid models not only obtain nearly lossless accuracies, but also exhibit significant advantages of low processing latency and low power consumption. In the next section, we will further analyze the energy consumption of different parts that are responsible for ANN calculations, SNN calculations, and signal conversions, respectively, in the hybrid models.

Analysis of Energy Consumption

Taking Model 5 to Model 8 as examples, we analyze the distribution of dynamic energy consumption and its change along with the time window. The dynamic energy distribution in these hybrid models when T_w equals 2, 4, 6, 8, and 10 are visualized in **Figure 8**. For each T_w value, the dynamic energy consumption transformation before and after the global adjustment of timing schedule are also plotted.

The Global adjustment of timing schedule can reduce the total dynamic energy consumption of network by reducing redundant ANN operations. Evidently, as the time window increases, the energy saving of the ANN calculations under the same conversion method retains the same. Model 5 and Model 7 use the spatial extension method and the input size of the ANN layer will increase along with T_w . As shown in **Figure 7**, the increase of the input size will consume more ANN FCores and thus more dynamic energy consumption. Nevertheless, due to the reduction of redundant operations brought by the global timing adjustment, the dynamic energy saving of ANN FCores in both Model 5 and Model 7 increases from about 50% to about 90%

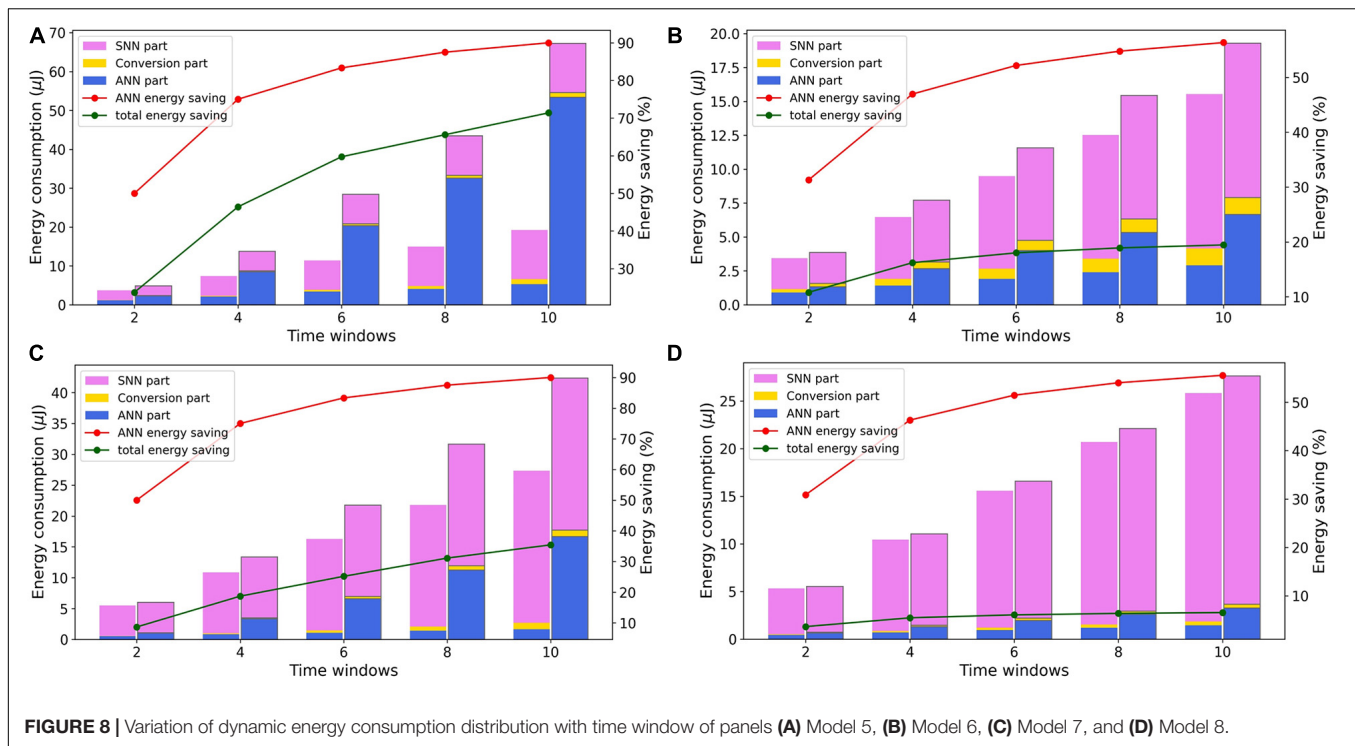


FIGURE 8 | Variation of dynamic energy consumption distribution with time window of panels (A) Model 5, (B) Model 6, (C) Model 7, and (D) Model 8.

as T_w increases. In Model 6 and Model 8, after the global timing adjustment, there are still some VMM operations that need to be executed repeatedly in the conversion FCores to drive the shift of the axon's spike buffer. Along with the increase of T_w , these repeated executions lead to the increase of ANN FCores' dynamic energy consumption in the adjusted models. As a result, Model 6 and Model 8 have a lower dynamic energy saving than Model 5 and Model 7, which increases from about 31% to about 56%. Additionally, due to the different proportion of ANN and SNN, the total energy saving also differs in different models. In a hybrid network, the larger the proportion of ANN calculations is, the more energy can be saved after global timing adjustment.

After the global timing adjustment, when T_w equals 2, the energy consumptions required for signal conversion in Model 5 to Model 8 accounts for a small part, which are 7.3%, 1.6%, 1.3%, and 0.9%, respectively. With the increase of T_w , these ratios all increase slightly. In the method of spatial expansion, because the number of conversion FCores is also increasing, the proportion will rise a little faster. But these ratios do not exceed 10% in the end. Furthermore, the difference in the dynamic energy consumed by SNN and ANN calculations is not as large as the difference in the number of FCores (about 6: 1 and 10: 1 in MLP and LeNet, respectively), indicating that the advantage of the computational sparsity in SNNs is well utilized in hardware execution.

CONCLUSION AND DISCUSSION

In this paper, we propose a systematic solution of implementing various hybrid networks on many-core neuromorphic

chips through software-hardware cooperation. Based on the abstraction of the Tianjic chip, we summarize that the fine-grained configurable basic units, unified communication format, and adjustable timing schedule provide the hardware foundation for implementation of hybrid models. On this basis, we propose an end-to-end mapping framework to facilitate implementation of hybrid models on hardware. By constructing basic connections for mixed dataflows, signal conversions are performed on the critical data paths of FCores without requiring additional devices. The configuration schemes for the typical four types of signal conversions are designed and proved to promote the mapping of the operations in hybrid models into FCores in the same way as that of the networks' computing operation. The global adjustment of timing schedule not only ensures the continuity and correctness of data transmission and processing in the network, but also reduces the energy consumption caused by the repeated redundant calculations. Furthermore, we built a tool chain to automatically implement the mapping framework. By mapping typical hybrid models to the Tianjic chip, we demonstrate that these models not only obtain almost lossless accuracy compared with the general computing platform, but also exhibit significant advantages of low execution latency and low power consumption. The results of the experiment show that although the implementation of signal conversion in HNNs generates additional resource overhead, the overall energy consumption of the network can be significantly reduced by disabling repeated redundant operations.

Generally, SNNs have rich coding schemes to encode information in spatio-temporal domain, including rate coding schemes (Deng et al., 2020b) and temporal coding schemes

like rank-order coding schemes (Tang et al., 2020), inter-spike interval based (Dong et al., 2019) and time-to-first spike based encoding schemes (Liu and Yue, 2017; Mostafa, 2017). Our framework can effectively support the rate coding and rank-order coding scheme for SNNs and in principle can support the temporal coding schemes via the unified communication format. The unified communication format can directly transmit the temporal information through its data segment (i.e., 8-bit *Fire_data* in the routing packet) to support the inter-spike interval based and time-to-first spike based encoding schemes. To effectively support these temporal coding schemes, efficient capture of the absolute or interval time information of each neuron's output spikes is necessary and requires further improved hardware design.

With the increasing complexity of tasks and the deepening of artificial intelligence research, it is expected that more and further cross-paradigm fusions of ANNs and SNNs will emerge. Hence, we are convinced that our proposed end-to-end hardware implementation method will provide a systematic solution to map hybrid models onto neuromorphic chips, and provide guidance for further development of hybrid neural models. Moreover, through the modeling abstraction of hardware characteristics, mapping mechanisms can be established to fully explore the potential of the hardware carrier and support more complex algorithm models. Through the iteration of hardware-software co-optimization, it is highly possible to develop a general brain-inspired computing platform that can handle more complex tasks.

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. Circ. Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396
- 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 *Proceedings of the 32nd Conference on Neural Information Processing System* Montreal.
- Chancán, M., Hernandez-Nunez, L., Narendra, A., Barron, A. B., and Milford, M. (2020). A hybrid compact neural architecture for visual place recognition. *IEEE Robot. Autom. Lett.* 5, 993–1000. doi: 10.1109/LRA.2020.2967324
- Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., et al. (2014). Dadiannao: a machine-learning supercomputer. *Paper Presented at the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (New Jersey, NJ: IEEE). doi: 10.1109/MICRO.2014.58
- Chen, Y.-H., Yang, T.-J., Emer, J., and Sze, V. (2019). Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* 9, 292–308. doi: 10.1109/JETCAS.2019.2910232
- 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
- Deng, L., Liang, L., Wang, G., Chang, L., Hu, X., Ma, X., et al. (2018). Semimap: a semi-folded convolution mapping for speed-overhead balance on crossbars. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* 39, 117–130. doi: 10.1109/TCAD.2018.2883959
- Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., et al. (2020a). Tianjic: a unified and scalable chip bridging spike-based and continuous neural computation. *IEEE J. Solid State Circ.* 55, 2228–2246. doi: 10.1109/JSSC.2020.2970709
- Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., et al. (2020b). Rethinking the performance comparison between SNNs and ANNs. *Neural Netw.* 121, 294–307. doi: 10.1016/j.neunet.2019.09.005
- Dong, S., Zhu, L., Xu, D., Tian, Y., and Huang, T. (2019). “An efficient coding method for spike camera using inter-spike intervals,” in *Proceedings of the 2019 Data Compression Conference* (New Jersey, NJ: IEEE). doi: 10.1109/DCC.2019.00080
- Esser, S. K., Andreopoulos, A., Appuswamy, R., Datta, P., Barch, D., Amir, A., et al. (2013). “Cognitive computing systems: algorithms and applications for networks of neurosynaptic cores,” in *Proceedings of the The 2013 International Joint Conference on Neural Networks* (New Jersey, NJ: IEEE). doi: 10.1109/IJCNN.2013.6706746
- 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. USA* 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
- Ghosh-Dastidar, S., and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308. doi: 10.1142/S0129065709002002
- Haessig, G., Cassidy, A., Alvarez, R., Benosman, R., and Orchard, G. (2018). Spiking optical flow for event-based sensors using IBM's truenorth neurosynaptic system. *IEEE Trans. Biomed. Circ. Syst.* 12, 860–870. doi: 10.1109/TBCAS.2018.2834558
- 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. *SIGARCH Comput. Archit. News* 44, 243–254. doi: 10.1145/3007787.3001163
- 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* (New Jersey, NJ: IEEE). doi: 10.1109/CVPR.2016.90
- Ji, Y., Zhang, Y., Chen, W., and Xie, Y. (2018). “Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler,” in

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

GW proposed the idea, designed and did the experiments, and wrote the manuscript. GW and YW conducted the algorithm modeling work. GW, SM, and JP conducted the design and implementation of the hardware testing platform. GW and SM contributed to the analysis and interpretation of results. RZ led the discussion and revised it. LS directed the project and provided overall guidance. All authors contributed to the article and approved the submitted version.

FUNDING

This work was partly supported by the National Key R&D Program of China 2018YFE0200200, National Nature Science Foundation of China (No. 61836004), Brain-Science Special Program of Beijing under grants Z181100001518006 and Z191100007519009, the Suzhou-Tsinghua innovation leading program 2016SZ0102, and CETC Haikang Group-Brain Inspired Computing Joint Research Center.

- Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA: ASPLOS). doi: 10.1145/3173162.3173205
- Jiao, Y., Han, L., Jin, R., Su, Y.-J., Ho, C., Yin, L., et al. (2020). "7.2 A 12nm programmable convolution-efficient neural-processing-unit chip achieving 825TOPS," in *Proceedings of the 2020 IEEE International Solid-State Circuits Conference-(ISSCC)* (New Jersey, NJ: IEEE). doi: 10.1109/ISSCC19947.2020.9062984
- 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.
- 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
- Lam, M. W., Chen, X., Hu, S., Yu, J., Liu, X., and Meng, H. (2019). Gaussian process Lstm recurrent neural network language models for speech recognition. *Paper Presented at the ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (New Jersey, NJ: IEEE). doi: 10.1109/ICASSP.2019.8683660
- Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521:436. doi: doi.org/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. doi: 10.1109/5.726791
- Lee, C., Kosta, A., Zhu, A. Z., Chaney, K., Daniilidis, K., and Roy, K. (2020). Spike-flownet: event-based optical flow estimation with energy-efficient hybrid neural networks. *[arXiv Preprint]* Available online at: <https://arxiv.org/abs/2003.06696> (accessed October 1, 2020).
- Liu, D., and Yue, S. (2017). Fast unsupervised learning for visual pattern recognition using spike timing dependent plasticity. *Neurocomputing* 249, 212–224. doi: 10.1016/j.neucom.2017.04.003
- 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
- Mahowald, M. (1993). *The Address-Event Representation Communication Protocol. AER 0.02*. (Pasadena, CA: California Institute of Technology).
- Marblestone, A. H., Wayne, G., and Kording, K. P. (2016). Toward an integration of deep learning and neuroscience. *Front. Comput. Neurosci.* 10:94. doi: 10.3389/fncom.2016.00094
- 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. (2017). 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
- 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
- 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
- Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., et al. (2019). "Simba: scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (New Jersey, NJ: IEEE).
- Shi, G., Liu, Z., Wang, X., Li, C. T., and Gu, X. (2017). Object-dependent sparse representation for extracellular spike detection. *Neurocomputing* 266, 674–686. doi: 10.1016/j.neucom.2017.05.082
- 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
- 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
- Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. (2017). Efficient processing of deep neural networks: a tutorial and survey. *Proc. IEEE* 105, 2295–2329. doi: 10.1109/JPROC.2017.2761740
- Tang, H., Cho, D., Lew, D., Kim, T., and Park, J. (2020). Rank order coding based spiking convolutional neural network architecture with energy-efficient membrane voltage updates. *Neurocomputing* 407, 300–312. doi: 10.1016/j.neucom.2020.05.031
- Ullman, S. (2019). Using neuroscience to develop artificial intelligence. *Science* 363, 692–693. doi: 10.1126/science.aau6595
- Wu, J., Wang, G., Yang, W., and Ji, X. (2016). Action recognition with joint attention on multi-level deep features. *[arXiv Preprint]* Available online at: <https://arxiv.org/abs/1607.02556> (accessed October 1, 2020).
- Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: faster, larger, better. *Paper Presented at the Proceedings of the AAAI Conference on Artificial Intelligence* (Menlo Park, CA: AAAI). doi: 10.1609/aaai.v33i01.33011311
- Yang, Y., Deng, L., Wu, S., Yan, T., Xie, Y., and Li, G. (2020). Training high-performance and large-scale deep neural networks with full 8-bit integers. *Neural Netw.* 125, 70–82. doi: 10.1016/j.neunet.2019.12.027
- Yang, Z., Wu, Y., Wang, G., Yang, Y., Li, G., Deng, L., et al. (2019). DashNet: a hybrid artificial and spiking neural network for high-speed object tracking. *[arXiv Preprint]* Available online at: <https://arxiv.org/abs/1909.12942> (accessed October 1, 2020).
- Zhang, B., Shi, L., and Song, S. (2016). Creating more intelligent robots through brain-inspired computing. *Sci. Robot.* 3:1445.

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.

Copyright © 2021 Wang, Ma, Wu, Pei, Zhao and Shi. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

Advantages of publishing in Frontiers



OPEN ACCESS

Articles are free to read
for greatest visibility
and readership



FAST PUBLICATION

Around 90 days
from submission
to decision



HIGH QUALITY PEER-REVIEW

Rigorous, collaborative,
and constructive
peer-review



TRANSPARENT PEER-REVIEW

Editors and reviewers
acknowledged by name
on published articles

Frontiers

Avenue du Tribunal-Fédéral 34
1005 Lausanne | Switzerland

Visit us: www.frontiersin.org

Contact us: frontiersin.org/about/contact



REPRODUCIBILITY OF RESEARCH

Support open data
and methods to enhance
research reproducibility



DIGITAL PUBLISHING

Articles designed
for optimal readership
across devices



FOLLOW US

@frontiersin



IMPACT METRICS

Advanced article metrics
track visibility across
digital media



EXTENSIVE PROMOTION

Marketing
and promotion
of impactful research



LOOP RESEARCH NETWORK

Our network
increases your
article's readership