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, resembling a neural network or a complex graph. The top half of the image has a solid blue background, while the bottom half is white.

BRAIN-INSPIRED COMPUTING: FROM NEUROSCIENCE TO NEUROMORPHIC ELECTRONICS DRIVING NEW FORMS OF ARTIFICIAL INTELLIGENCE

EDITED BY: Jonathan Mapelli, Giacomo Indiveri and Angelo Arleo

PUBLISHED IN: *Frontiers in Neuroscience*, *Frontiers in Cellular Neuroscience*
and *Frontiers in Computational 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 978-2-88974-608-8

DOI 10.3389/978-2-88974-608-8

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

BRAIN-INSPIRED COMPUTING: FROM NEUROSCIENCE TO NEUROMORPHIC ELECTRONICS DRIVING NEW FORMS OF ARTIFICIAL INTELLIGENCE

Topic Editors:

Jonathan Mapelli, University of Modena and Reggio Emilia, Italy

Giacomo Indiveri, University of Zurich, Switzerland

Angelo Arleo, Centre National de la Recherche Scientifique (CNRS), France

Citation: Mapelli, J., Indiveri, G., Arleo, A., eds. (2022). Brain-Inspired Computing: From Neuroscience to Neuromorphic Electronics Driving New Forms of Artificial Intelligence. Lausanne: Frontiers Media SA. doi: 10.3389/978-2-88974-608-8

Table of Contents

- 04 *Optimizing BCPNN Learning Rule for Memory Access***
Yu Yang, Dimitrios Stathis, Rodolfo Jordão, Ahmed Hemani and Anders Lansner
- 19 *Neuromorphic Computing Using NAND Flash Memory Architecture With Pulse Width Modulation Scheme***
Sung-Tae Lee and Jong-Ho Lee
- 29 *Unsupervised Adaptive Weight Pruning for Energy-Efficient Neuromorphic Systems***
Wenzhe Guo, Mohammed E. Fouda, Hasan Erdem Yantir, Ahmed M. Eltawil and Khaled Nabil Salama
- 47 *Efficient Spike-Driven Learning With Dendritic Event-Based Processing***
Shuangming Yang, Tian Gao, Jiang Wang, Bin Deng, Benjamin Lansdell and Bernabe Linares-Barranco
- 62 *Neuromorphic Analog Implementation of Neural Engineering Framework-Inspired Spiking Neuron for High-Dimensional Representation***
Avi Hazan and Elishai Ezra Tsur
- 73 *Real Time Generation of Three Dimensional Patterns for Multiphoton Stimulation***
Paolo Pozzi and Jonathan Mapelli
- 83 *Hardware Design for Autonomous Bayesian Networks***
Rafatul Faria, Jan Kaiser, Kerem Y. Camsari and Supriyo Datta
- 93 *Oscillatory Neural Networks Using VO₂ Based Phase Encoded Logic***
Juan Núñez, María J. Avedillo, Manuel Jiménez, José M. Quintana, Aida Todri-Sanial, Elisabetta Corti, Siegfried Karg and Bernabé Linares-Barranco
- 102 *Event-Based Trajectory Prediction Using Spiking Neural Networks***
Guillaume Debat, Tushar Chauhan, Benoit R. Cottureau, Timothée Masquelier, Michel Paindavoine and Robin Baures
- 118 *Understanding the Impact of Neural Variations and Random Connections on Inference***
Yuan Zeng, Zubayer Ibne Ferdous, Weixiang Zhang, Mufan Xu, Anlan Yu, Drew Patel, Valentin Post, Xiaochen Guo, Yevgeny Berdichevsky and Zhiyuan Yan
- 131 *Quantifying the Brain Predictivity of Artificial Neural Networks With Nonlinear Response Mapping***
Aditi Anand, Sanchari Sen and Kaushik Roy



Optimizing BCPNN Learning Rule for Memory Access

Yu Yang^{1*}, Dimitrios Stathis¹, Rodolfo Jordão¹, Ahmed Hemani¹ and Anders Lansner^{2,3}

¹ Division of Electronics and Embedded Systems, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, ² Division of Computational Science and Technology, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, ³ Department of Mathematics, Stockholm University, Stockholm, Sweden

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

James Courtney Knight,
University of Sussex, United Kingdom
Francesco Maria Puglisi,
University of Modena and Reggio
Emilia, Italy

*Correspondence:

Yu Yang
yuyang2@kth.se

Specialty section:

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

Received: 13 June 2020

Accepted: 28 July 2020

Published: 31 August 2020

Citation:

Yang Y, Stathis D, Jordão R, Hemani A
and Lansner A (2020) Optimizing
BCPNN Learning Rule for Memory
Access. *Front. Neurosci.* 14:878.
doi: 10.3389/fnins.2020.00878

Simulation of large scale biologically plausible spiking neural networks, e.g., Bayesian Confidence Propagation Neural Network (BCPNN), usually requires high-performance supercomputers with dedicated accelerators, such as GPUs, FPGAs, or even Application-Specific Integrated Circuits (ASICs). Almost all of these computers are based on the von Neumann architecture that separates storage and computation. In all these solutions, memory access is the dominant cost even for highly customized computation and memory architecture, such as ASICs. In this paper, we propose an optimization technique that can make the BCPNN simulation memory access friendly by avoiding a dual-access pattern. The BCPNN synaptic traces and weights are organized as matrices accessed both row-wise and column-wise. Accessing data stored in DRAM with a dual-access pattern is extremely expensive. A post-synaptic history buffer and an approximation function thus are introduced to eliminate the troublesome column update. The error analysis combining theoretical analysis and experiments suggests that the probability of introducing intolerable errors by such optimization can be bounded to a very small number, which makes it almost negligible. Derivation and validation of such a bound is the core contribution of this paper. Experiments on a GPU platform shows that compared to the previously reported baseline simulation strategy, the proposed optimization technique reduces the storage requirement by 33%, the global memory access demand by more than 27% and DRAM access rate by more than 5%; the latency of updating synaptic traces decreases by roughly 50%. Compared with the other similar optimization technique reported in the literature, our method clearly shows considerably better results. Although the BCPNN is used as the targeted neural network model, the proposed optimization method can be applied to other artificial neural network models based on a Hebbian learning rule.

Keywords: Bayesian Confidence Propagation Neural Network (BCPNN), neuromorphic computing, Hebbian learning, spiking neural networks, memory optimization, DRAM, cache, digital neuromorphic hardware

1. INTRODUCTION

Bayesian Confidence Propagation Neural Networks (BCPNNs), proposed by Lansner and Ekeberg (1989) and Lansner and Holst (1996), are biologically plausible brain cortex models that have been proven useful for understanding brain functions. Tully et al. (2016) implemented a BCPNN on SpiNNaker and analyzed the neural structure and dynamics inside a hypercolumn

and demonstrated temporal sequence learning. Meli and Lansner (2013) studied the neural interconnection scheme from a BCPNN model. Fiebig et al. (2020) demonstrated how BCPNN could emulate the cortical working memory function. Recently, unsupervised hidden representation learning using BCPNN was benchmarked on MNIST. The BCPNN achieved 97.5% accuracy on the unseen test set (Ravichandran et al., 2020).

Currently, the simulation of large scale BCPNNs heavily relies on high-performance computing centers equipped with supercomputers and accelerators, such as GPUs and ASICs (Farahini et al., 2014; Stathis et al., 2020), or dedicated spiking neural network simulation platform, such as SpiNNaker (Knight et al., 2016). We identify three categories of optimization methods: (1) reducing the amount of computation, (2) reducing the amount of memory access demand, and (3) increasing the memory access efficiency. Current studies of the BCPNN optimization are mainly focused on reducing the computation and memory access demand (Vogginger et al., 2015). The memory access efficiency aspect is seldom exploited.

With technology scaling, memory access becomes the dominant cost for most applications (Mutlu, 2013). Memory optimization in terms of both reducing memory access demand and increasing the efficiency of the memory access has been done for many years both for non-Hebbian artificial neural networks and Hebbian spiking neural networks. For conventional non-spiking deep neural networks, research works like Li et al. (2016) and Yang et al. (2017) optimized both memory access demand and efficiency in deep convolutional neural networks. For Hebbian spiking neural networks, most research works target the spike-timing-dependent plasticity (STDP) learning rule (Markram et al., 2012). For example, Bichler et al. (2012) simplified the STDP learning rule and reduced the demand for computation and memory access. Yousefzadeh et al. (2017) further improved the method proposed by Bichler et al. by replacing the full connection to a weight-sharing connection. Thus, it further reduced the computation and memory access demand. Davies et al. (2012) changed part of the STDP learning rule and approximated the membrane potential in LTP. It reduced the computation and memory access demand and improved memory access efficiency. Jin et al. (2010) and Davies et al. (2018) delayed the update of weights and reduced the memory access demand. Pedroni et al. (2019) analyzed different synaptic matrix memory mapping strategies and proposed a variation of STDP learning rule to perform the causal and acausal update process. It reduced memory storage demand for the sparsely connected network by pointer-based compressed sparse rows and improved the efficiency for reversed access of such pointer-based data structure. Knight and Furber (2016) proposed a spike buffer and a mechanism called “flushing event” to deal with the inefficient column update in STDP and increase the memory efficiency. Morrison et al. (2007) used a dynamic spike buffer to remove column update process in STDP and increased memory efficiency. Sheik et al. (2016) also pointed out the memory problem caused by bi-directional spike-triggered learning rule and proposed a learning rule in which update is only triggered by presynaptic spikes to improve the memory access efficiency. However, almost all of these studies that target

spiking neural networks focus on the STDP learning rule. Thus, it cannot be directly applied to BCPNN since its learning rule is different and more complex than STDP. By abandoning a conventional von Neumann architecture, custom neural network simulation platforms could potentially avoid the root of the memory access problems. Serrano-Gotarredona et al. (2013) and Prezioso et al. (2018) used memristors to merge computation and storage, thus eliminating the need for memory access. Though such new architectures are efficient and attractive, they are not off-the-shelf and easily accessible, and none of them supports the BCPNN learning rule.

In this paper, we tackle the memory access problem introduced by the BCPNN optimization method presented in Vogginger et al. (2015). By replacing a time-driven simulation method with an event-driven one, plenty of computational requirements have been eliminated. The event-driven simulation method is called “lazy evaluation method” because it delays the evaluation computation as much as possible. The lazy evaluation simulation method requires access to the synaptic matrix stored in main memory, both row-wise and column-wise. A presynaptic spike triggers an update of a single row in the synaptic matrix, while a post-synaptic spike triggers an update of a single column. Today’s memory architecture, such as DRAM, cannot handle two orthogonal directional access patterns of the same block of continuous data without sacrificing efficiency. Such access patterns will also affect the efficiency of the cache system. Therefore, we propose to remove the column update procedure and to merge the column and row update. In this way, we avoid entirely the dual memory access pattern that degrades the efficiency of the BCPNN simulation. Furthermore, by carefully designing our strategy, we can also reduce the demand in storage requirement and memory access, while increasing the overall performance. We remind readers that even though the BCPNN is the optimization target, our method is not restricted to this learning rule. Any Hebbian based neural network learning rule could potentially be optimized with a slightly modified version of our strategy.

The rest of the paper is organized as follows: section 2 explains the original BCPNN learning rule, points out the memory access problem, proposes the alternative method that resolves the problem, and performs an error analysis for the proposed method. Section 3 demonstrates the benefits of the proposed method in terms of both memory storage requirements and performance. Finally, section 4 summarizes the paper and addresses the potential of the proposed method.

2. METHODOLOGY

In this section, we introduce the lazy evaluation BCPNN simulation strategy and highlight the memory access problem. To overcome this problem, we propose an optimization technique that tackles the memory access problem. We also present a detailed analytical and experimental error analysis that shows the probability of introducing intolerable errors is negligible. Since we will not drastically modify the BCPNN learning rule in this paper, we will just present its essence. Readers can

find the complete and detailed description in Vogginger et al. (2015).

2.1. The BCPNN Learning Rule

BCPNN is a type of artificial neural network whose learning rule is derived from Bayes' theorem. It strengthens or weakens the connectivity/weight between pre- and post-synaptic neurons based on their co-activation. A correlated pre- and post-synaptic activity gives a positive weight, whereas an anti-correlation gives a negative weight. The connectivity/weight is calculated by Bayes' theorem based on the measured firing probability of pre- and post-synaptic neurons.

To mimic the biological columnar cortical structure (Buxhoeveden and Casanova, 2002), BCPNN considers minicolumn units (MCUs) as its basic units representing the aggregation of about a hundred neurons. Many MCUs form a hypercolumn unit (HCU) representing the biological hypercolumn structure (Hubel and Wiesel, 1974). MCUs in each HCU compete with each other in a soft winner-take-all (soft WTA) fashion, representing the net effects of excitatory and inhibitory connections among neural cells, as shown in Coultrip et al. (1992) and Lundqvist et al. (2006). We use the notation $H \times M$ to represent a BCPNN configuration that consists of H HCUs, and each HCU includes M MCUs. Usually, we have the constraint $H \geq M$. H can be increased arbitrarily without an upper limit. M , on the other hand, has a limit of $M = 100$. Therefore, the network is growing purely due to the growth of the amount of HCUs for big networks. For example, a typical human cortex comparable BCPNN configuration is $\sim 2 \cdot 10^6 \times 100$ (Johansson and Lansner, 2007).

Each MCU could connect to an HCU via a sparse patchy connection (Meli and Lansner, 2013). Synapses are formed due to these connections. Fully connected big networks are very costly in terms of storage and computation. A parameter C constrains the amount of possible incoming connection slots of each HCU. The parameters C and M define the shape of its synaptic matrix. In each human cortex comparable HCU, a $10^4 \times 100$ synaptic matrix is used to store the intermediate synaptic traces as well as the synaptic weights, as shown in Figure 1A. This HCU configuration uses $C = 10^4$ incoming connections and $M = 100$ MCUs. On the presynaptic side (left side) of the synaptic matrix, an i -vector of size $C = 10^4$ is used to store presynaptic traces z_i , e_i , and p_i . On the post-synaptic side (bottom side), a j -vector of size $M = 100$ is used to store post-synaptic traces z_j , e_j , and p_j . The synaptic matrix is also called the ij -matrix because it stores the synaptic traces e_{ij} , p_{ij} , and w_{ij} .

Following Bayes' theorem, the BCPNN learning rule requires the probability estimation of both pre- and post-synaptic events (spikes). Such probability estimation is obtained via a chain of low-pass filters applied on the pre- and post-synaptic spikes (s_i and s_j). For example, the presynaptic filter chain is $s_i \rightarrow z_i \rightarrow e_i \rightarrow p_i$. Similar filter chains are also used to generate p_j ($s_j \rightarrow z_j \rightarrow e_j \rightarrow p_j$) and p_{ij} ($z_i * z_j \rightarrow e_{ij} \rightarrow p_{ij}$). The last three traces in the chain (p_i , p_j , and p_{ij}) represent the firing probability for presynaptic spikes, post-synaptic spikes as well as pre- and post-synaptic spike co-activation. The synaptic weight is then computed by combining these three traces. These filters

are implemented by ordinary differential equations (ODEs). Equation (1) is an example of such ODEs where X is the input, Y is the output, and τ is the time constant. These ODEs can be easily computed by Euler's method (Griffiths and Higham, 2010) in time-driven simulation.

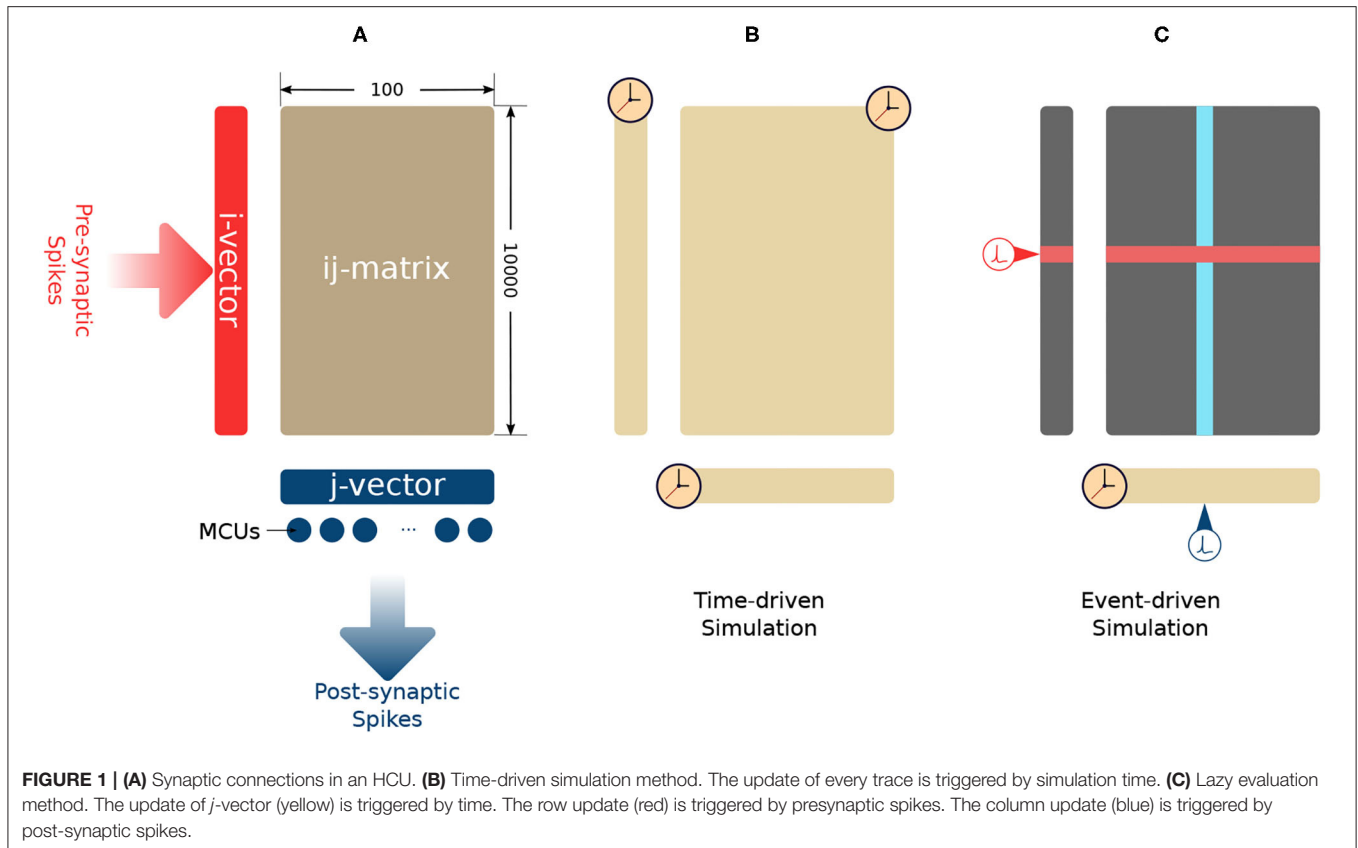
$$\tau \frac{dY}{dt} = X - Y \quad (1)$$

As a numeric method, Euler's method requires updating all the traces whenever the simulation time is forwarded one simulation step Δt , as shown in Figure 1B. The synaptic plasticity is caused by the interaction between pre- and post-synaptic spikes. These spikes drive the change of synaptic weights. The update of synaptic traces is not necessary when the spikes are absent. As long as we account for the decay of all the traces when a pre- or post-synaptic spike comes, then the network's mathematical behavior will be the same as for time-driven simulation. As shown in Figure 1C, a lazy evaluation method only updates part of i -vector and ij -matrix when there is a triggering spike. The current value of traces will be calculated analytically based on the time difference between the current simulation step and the time when the previous spike came. In this paper, we use the "Analytical I method" in Vogginger et al. (2015) as the baseline and refer it as the "lazy evaluation method." Equation (2) shows the complete set of equations of the lazy evaluation method. These equations are used for calculating p_i and p_{ij} . The calculation of p_j remains as time-driven. The detail of the lazy evaluation method is not in the scope of this paper, readers can find the proof of equivalence in Vogginger et al. (2015).

$$\begin{aligned} z_i(t) &= z_i(t^{last}) \cdot e^{-\frac{\Delta t}{\tau_{z_i}}} + s_i(t) \\ e_i(t) &= e_i(t^{last}) \cdot e^{-\frac{\Delta t}{\tau_e}} + a_i \left(e^{-\frac{\Delta t}{\tau_{z_i}}} - e^{-\frac{\Delta t}{\tau_e}} \right) z_i(t^{last}) \\ p_i(t) &= p_i(t^{last}) \cdot e^{-\frac{\Delta t}{\tau_p^*}} + a_i b_i \left(e^{-\frac{\Delta t}{\tau_{z_i}}} - e^{-\frac{\Delta t}{\tau_p^*}} \right) z_i(t^{last}) \\ &\quad + \left(e_i(t^{last}) - a_i z_i(t^{last}) \right) c \left(e^{-\frac{\Delta t}{\tau_e}} - e^{-\frac{\Delta t}{\tau_p^*}} \right) \\ e_{ij}(t) &= e_{ij}(t^{last}) \cdot e^{-\frac{\Delta t}{\tau_e}} + a_{ij} \left(e^{-\frac{\Delta t}{\tau_{z_{ij}}}} - e^{-\frac{\Delta t}{\tau_e}} \right) z_i(t^{last}) z_j(t^{last}) \\ p_{ij}(t) &= p_{ij}(t^{last}) \cdot e^{-\frac{\Delta t}{\tau_p^*}} + a_{ij} b_{ij} \left(e^{-\frac{\Delta t}{\tau_{z_{ij}}}} - e^{-\frac{\Delta t}{\tau_p^*}} \right) z_i(t^{last}) z_j(t^{last}) \\ &\quad + \left(e_{ij}(t^{last}) - a_{ij} z_i(t^{last}) z_j(t^{last}) \right) c \left(e^{-\frac{\Delta t}{\tau_e}} - e^{-\frac{\Delta t}{\tau_p^*}} \right) \end{aligned} \quad (2)$$

where,

$$\begin{aligned} a_i &= \frac{\tau_{z_i}}{\tau_{z_i} - \tau_e} & b_i &= \frac{\tau_{z_i}}{\tau_{z_i} - \tau_p^*} & c &= \frac{\tau_e}{\tau_e - \tau_p^*} \\ \tau_{z_{ij}} &= \left(\frac{1}{\tau_{z_i}} + \frac{1}{\tau_{z_j}} \right)^{-1} & a_{ij} &= \frac{\tau_{z_{ij}}}{\tau_{z_{ij}} - \tau_e} & b_{ij} &= \frac{\tau_{z_{ij}}}{\tau_{z_{ij}} - \tau_p^*} \end{aligned}$$



Each MCU works as a leaky integrator that integrates the input spike effects ($s_i \cdot w_{ij}$) in terms of membrane potential. These MCUs in the same HCU then compete with each other based on their membrane potential in soft-WTA fashion. The soft-WTA process normalizes their membrane potential and generates a relative firing rate o_j . If the soft-WTA process has selected a winning MCU, the o_j of the winning MCU will be approaching 1. The rest losing MCUs will be suppressed to an o_j approaching 0. That means the winning MCU will have a higher probability of firing than the rest. If there is no clear winner, all the MCUs will have nearly uniform o_j after the soft-WTA process. In this case, the MCUs will have a relatively low but equal firing probability. The o_j is then scaled to match the firing rate range in order to give the final firing rate r_j , Equation (3). The firing rate range is between 0 and the maximum firing rate (r_{max}), where the maximum firing rate is usually set to 0.1.

$$r_j = r_{max} \cdot o_j \quad (3)$$

Finally, to generate a spike from the firing rate r_j , a Poisson spike generator (Dayan and Abbott, 2001) shown in Equation (4) is employed. A uniformly distributed random number x is generated every time and compared to r_j . If r_j is bigger than x , the MCU fires.

$$s_j = \begin{cases} 1, & \text{if } r_j > x, x \sim \mathcal{U}(0, 1) \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

2.2. Memory Access Problems of Lazy Evaluation Method

By adopting the lazy evaluation method, the massive memory access and computation demand from the time-driven simulation can now be avoided. However, the lazy evaluation method is not perfect. Its most significant issue is the dual-memory access pattern. Both pre- and post-synaptic spikes trigger the update events of synaptic traces. Depending on the type of spikes, either a row or a column of the synaptic matrix is fetched. The row or column is sent to the computation unit to be updated and stored back. The lazy evaluation implies that memory storage should efficiently support both row-wise and column-wise access mode to achieve high throughput. However, such a requirement is not feasible for modern DRAM and cache hierarchy of supercomputers.

Modern DRAM and cache respond to every *READ* and *WRITE* operation with a whole row of their data to increase efficiency. Isolated single cell access pattern is not friendly for DRAM and cache. To access a single data cell, they need to fetch

or store a whole row of data. Since other parts of the row are useless, to operate on them is a waste of time and energy.

The lazy evaluation method with both row-wise and column-wise access patterns can be a great challenge for the DRAM and cache system. In Stathis et al. (2020), the author tries to customize the DRAM architecture to make it more adaptable to the BCPNN access pattern. However, due to the nature of DRAM technology, even by heavy customization, it still sacrifices the DRAM performance to support lazy evaluation. In this work, we propose a modified lazy evaluation method that eliminates the column-wise memory access pattern. This optimization makes the BCPNN learning rule DRAM and cache access friendly. We refer to the modified algorithm as *Column Update Elimination*, or *CUE* for short.

The CUE method not only solves the dual memory access problem but also eliminates the demand for memory access by the column update process. Even with the lazy evaluation method, which has already dramatically reduced the memory access, the memory access demand is still huge for a large BCPNN. To put it into context, a human cortex comparable BCPNN has a $2 \cdot 10^6 \times 100$ configuration. Such a network needs to access, on average, 10,000 rows and 100 columns of synaptic storage in every second per HCU. In total, it will require 200 TB of data traffic for the whole network per second. The amount of data required by 100 rows and 1 column is the same, around 240 KB/HCU, because the shape of the synaptic matrix is $10,000 \times 100$ for the human cortex comparable BCPNN. By applying CUE, half of the memory access demand due to column update is eliminated.

2.3. Column Update Elimination (CUE)

In this section, we modify the original BCPNN lazy evaluation method by eliminating the more expensive column update. In each subsection, we will introduce one important modification and explain in detail its mechanism. The main idea of this modification is to avoid column-wise access to DRAM and cache by removing the column update. The row update triggered by the presynaptic spike updates the synaptic weight, which directly influences the spike generation. On the other hand, the column update only affects the state of the synaptic traces and can be delayed. The column update can be removed, and its calculations can be integrated with the row update.

2.3.1. Ideal CUE

As shown in **Figure 2A**, in the lazy evaluation method, each cell in the *ij*-matrix will be updated either by row update (red) or column update (blue). A spike triggers each update event. The update procedure includes three tasks: Load data from memory, perform the computation, and store data back to memory. Row update is triggered by presynaptic spikes s_i (red), while column update is triggered by post-synaptic spikes s_j (blue). The example in **Figure 2A**, shows the update event of a single cell in the *ij*-matrix. From left to right, its traces are updated by a row update, followed by 3 column updates, and another row update. Each update only covers the range from its trigger point back until the last spike event.

If an infinite buffer that records all the post-synaptic spikes is available, the column update can then be eliminated, as shown in

Figure 2B. Each cell in the *ij*-matrix will only be updated by a row update (red). The row update will cover the range from its trigger point back until the last presynaptic spike event. However, the row update process in **Figure 2B** is different from **Figure 2A**. The new row update process emulates the computation of the original column updates thanks to the buffer that keeps the record of all post-synaptic spikes. Compared to the original lazy evaluation method, the CUE row update reduces the amount of memory access leaving only the row-wise memory access patterns.

The CUE method with an infinite sized buffer will not introduce any additional error, as all the computation required by the lazy evaluation method is still performed. It only changes the point in time when each computation happens. The CUE does not reduce the amount of computation. It only reduces the amount of memory access. The integration of column update effects is summarized by Algorithms 1 and 2.

Algorithm 1: Lazy Evaluation Method

```

while a presynaptic spike arrives do
  foreach Cell ∈ Row do
    load_memory(Cell);
    update_traces(Cell);
    store_memory(Cell);
  end
end
while a post-synaptic spike is generated do
  foreach Cell ∈ Column do
    load_memory(Cell);
    update_traces(Cell);
    store_memory(Cell);
  end
end

```

Algorithm 2: Ideal CUE Method

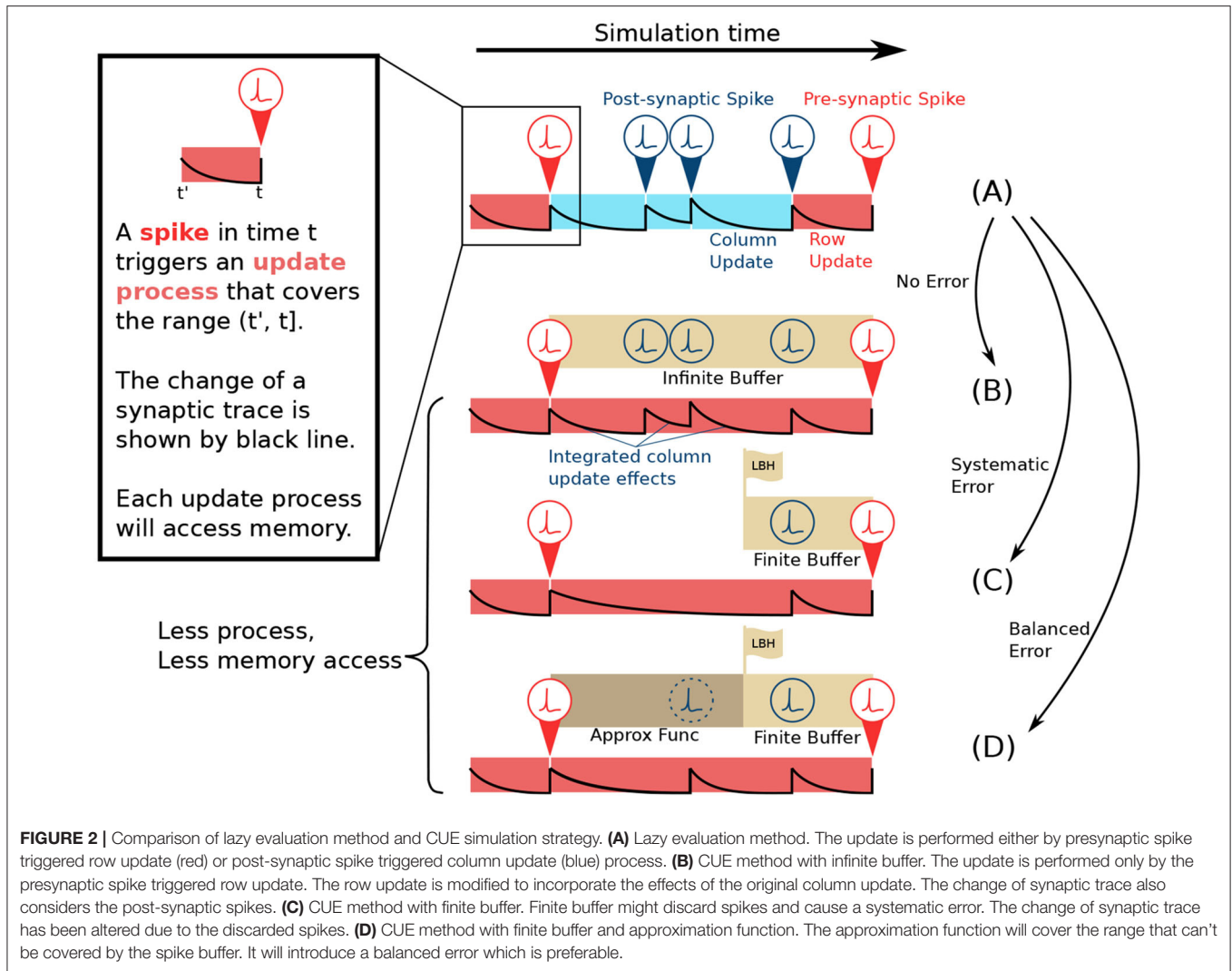
```

while a presynaptic spike arrives do
  foreach Cell ∈ Row do
    load_memory(Cell);
    // Integrating column update by the following
    FOR-LOOP;
    foreach post-synaptic spike ∈ Buffer[Cell.Column] do
      update_traces(Cell);
    end
    update_traces(Cell);
    store_memory(Cell);
  end
end

```

2.3.2. CUE With Finite Sized Buffer

To have an infinite buffer that records every post-synaptic spike is impractical. In a more realistic case, the infinite buffer can be substituted by one with limited size, as shown in **Figure 2C**. The



edge of the buffer is called the look-back horizon (LBH). A spike history buffer with size L can only respond to the request for $s_j(t)$ when t is later in time than the LBH. The information from spike events that are earlier in time than LBH is lost.

As shown in the example in **Figure 2C**, two post-synaptic spikes that are beyond the LBH are discarded by the spike history buffer, due to its limited size. Therefore, the row update computation, that emulates the effects of the intermediate column updates is different from the original lazy evaluation computation. Such behavior will introduce systemic errors since spikes are solely dropped. The post-synaptic spike train observed by the row update in **Figure 2C** will always have fewer spikes than the spike train in **Figure 2A**. This systematic error is undesirable because it will accumulate strictly positively or negatively. We need a mechanism that could introduce balanced errors so that they can potentially cancel each other.

2.3.3. Approximation Function

To avoid a systematic error, we propose an approximation function to predict the spikes beyond the LBH, as shown in

Figure 2D. Since the spike history is lost, due to the limited buffer size, the prediction is the only option when the information of $s_j(t)$ beyond LBH is needed. An approximation function is defined as $\mathcal{H}: \mathbb{N}^0 \times \mathbb{D} \rightarrow \mathbb{B}; (t, m) \mapsto s$, where $t \in \mathbb{N}^0$ is the simulation step that needs prediction, $m \in \mathbb{D}$ is whatever extra information required by the approximation function, and $s \in \mathbb{B}$ is the boolean variable indicating whether a spike should be generated or not.

Errors will be introduced if the approximation function is not an oracle that always gives a correct prediction. In the next section, we will discuss various approximation functions in detail and analyze their error bounds. A good approximation function should be computationally light and be able to reduce the error, compared to the scenario when all the spikes beyond LBH are dropped. Any errors introduced by the approximation function should be introduced in a balanced way to avoid error accumulation.

2.3.4. Alternative Approach in Literature

Knight et al. (2016) has reported a similar method to remove the BCPNN column update by introducing a finite buffer. The

work does not mention how the expired spikes are handled. If the expired spikes are just dropped, the BCPNN will suffer from systematic error.

Another paper by the same author (Knight and Furber, 2016) has reported a “flushing event” mechanism to avoid the loss of spikes when using a finite-sized buffer for STDP. The flushing event method used in STDP is not very friendly for BCPNN. A presynaptic neuron triggers a flushing event when it hasn’t been active for a fixed time (usually spike buffer size L) and triggers an update of a row in the synaptic matrix. In BCPNN, when the internal representation is stable, the active rows (very small proportion of the synaptic matrix) are also stable. For example, in a 100×100 network with 10,000 rows in total, the number of active rows is statistically always the same 100 rows due to the bursty property of spike trains. Therefore, presynaptic neuron triggered flushing events will force almost a complete full matrix update every L ms. In this paper, we have optimized the flushing event method to be triggered by post-synaptic spikes. It forces a column update whenever a post-synaptic spike is shifted out from the spike buffer to guarantee that no spike is lost. The post-synaptic triggered version statistically only updates a synaptic column instead of the whole synaptic matrix every L ms. Different from our approach which uses approximation function to predict the spikes, the flushing event method is exact and will not introduce error. But the price would be keeping the inefficient column update process.

In section 3, we implement the flushing event method on BCPNN and compare both the flushing event method and our CUE method against the baseline lazy evaluation method. Readers will see that by removing the column update process, CUE method outperforms the flushing event method in terms of both storage and performance.

2.4. Error Analysis

In this section, we analyze the error introduced by the CUE method. The error discussed in this paper refers to the relative error of synaptic weight w_{ij} caused by wrong post-synaptic spike predictions. We choose w_{ij} because it is the final synaptic variable that influences the spike generation. The error is treated only at the evaluation points since the row update only happens at these timing points. The error is defined by Equation (5). For simplicity, we denote the evaluation of $err(t)$ at the evaluation points simply by err in later text. In this section, we bound the probability of intolerable error. Two small threshold numbers, ϵ and δ , are defined for such error bound. Equation (6) describes the err , as a function of ϵ and δ .

$$err(t) = \begin{cases} \left| \frac{w_{ij}(t) - w_{ij}^{pred}(t)}{w_{ij}(t)} \right|, & \text{if } s_i(t) = 1 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$P(err > \epsilon | L, \mathcal{H}) = P(t - t' > L) \cdot P(err > \epsilon | \mathcal{H}) \leq \delta \quad (6)$$

Equation (6) represents the probability of having an error that exceeds the threshold ϵ , under the scenario that a spike buffer

of size L is used together with an approximation function \mathcal{H} . This probability is bounded by δ . If we prove that δ is sufficiently small, we can assert that the BCPNN behavior will not diverge from the original simulation strategy. We do not intend to mathematically bound the error to a definite number that works for every corner case because such an approach is very pessimistic and does not reflect typical operational scenarios.

The probability $P(err > \epsilon | L, \mathcal{H})$ can be expanded as two terms. The two terms describe the conditions: (1) The last presynaptic spike fires beyond the look-back horizon (LBH), and (2) the approximation function prediction gives an intolerable error. In Equation (6), t is the current evaluation time when a presynaptic spike is observed, and t' is the time when the last presynaptic spike occurs for the same synaptic cell.

In the following sections, we first discuss the spike firing rate distribution to form the foundation for the probability-bound calculation. Then we calculate the probability of a presynaptic spike firing beyond the LBH, $P(t - t' > L)$. After that, we present two approximate functions—the static and adaptive approximation function. We establish the probability of introducing errors via the two approximation functions, $P(err > \epsilon | \mathcal{H})$. Finally, we summarize all the strategies and compute the overall error probability bound.

2.4.1. Spike Firing Rate Distribution

The average spiking frequency in higher-order (memory/cognitive) cortical areas is very low, likely around 0.1 Hz (Lennie, 2003). When experimenters record active neurons, they typically have spiking frequency up to 100 Hz with an average of around 20 Hz. An MCU in BCPNN does not directly represent every single neuron. Instead, it mimics the behavior of a minicolumn of some hundred neurons, of which, only a handful (5–10) big layer 5 pyramidal cells communicate outside the HCU. So we estimate that the maximum instantaneous firing frequency of an MCU is $5 \times 20 = 100$ Hz. Therefore, the maximum firing rate at each simulation step r_{max} is set to 0.1 when the simulation step Δt is set to 1 ms.

An $H \times M$ BCPNN has an activity level $\alpha \in [0, 1]$. It indicates the number of active HCUs in this BCPNN is αH . An active HCU is an HCU with a clear winning MCU and some losing MCUs after the soft Winner-Take-All (soft WTA) process while an inactive HCU has only MCUs that are neither winners nor losers, see section 2.1. Winning MCUs will have a high firing rate (close to r_{max}), losing MCUs have a low firing rate (close to 0), and MCUs in the inactive HCU will have uniform firing rate (close to $\frac{r_{max}}{M}$). Usually, a BCPNN should not have the majority of its HCUs inactive as those HCUs would not be able to learn.

The first row of **Figures 3A–C**, show examples of measured firing rate probability distribution function $\rho(r)$ of a 10×10 BCPNN. From the figure, we can see that the distribution of firing rates is very dense in some extremely narrow areas. If we zoom in, as shown in sub-figure (C), we can observe a narrow bell-curve shaped distribution. The cause of each peak is marked by text beside all the sub-figures in the first row (A), (B), and (C). For example, sub-figure (A) has only one peak caused by the MCUs in the inactive HCUs, since when $\alpha = 0$, all HCUs are inactive.

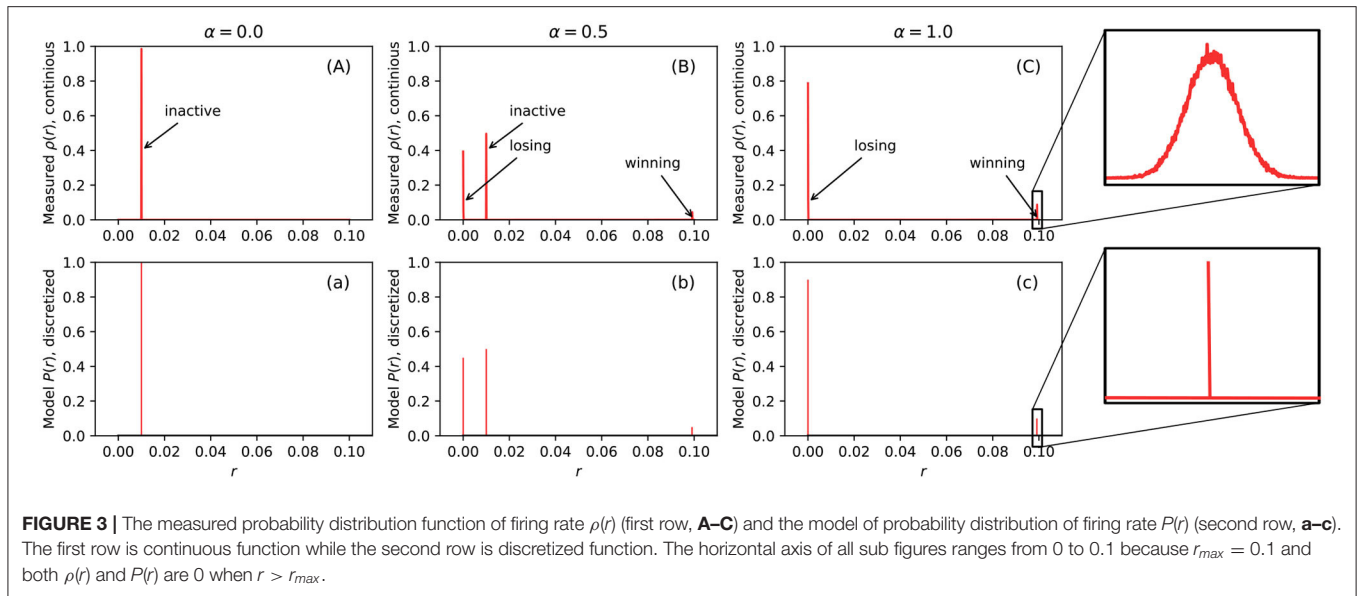


FIGURE 3 | The measured probability distribution function of firing rate $\rho(r)$ (first row, **A–C**) and the model of probability distribution of firing rate $P(r)$ (second row, **a–c**). The first row is continuous function while the second row is discretized function. The horizontal axis of all sub figures ranges from 0 to 0.1 because $r_{max} = 0.1$ and both $\rho(r)$ and $P(r)$ are 0 when $r > r_{max}$.

In sub-figure (C), two peaks are caused by losing and winning MCUs inside the active HCUs, since when $\alpha = 1$, all HCUs are active, hence there is a winning MCU and some losing MCUs. At last, when $\alpha = 0.5$, it is the combination of the previous two scenarios. The height of each pulse is determined by the amount of MCUs that cause it. If we compare the peak height caused by losing and winning MCUs, we can see that the losing peak is much higher than the winning one, because there are nine times more losing MCUs than the winning MCUs in active HCUs due to the soft-WTA process.

$$P(r) = \begin{cases} \alpha \frac{M-1}{M}, & \text{if } r = r_l, \\ \alpha \frac{1}{M}, & \text{if } r = r_w, \\ 1 - \alpha, & \text{if } r = r_s, \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

We can use impulse functions $\delta(\cdot)$ to model the probability distribution function (PDF) of firing rate $\rho(r)$. By using a δ function, the probability distribution of the firing rate is discretized. Therefore, we can directly use $P(r)$ shown in Equation (7) to describe the discretized events. $P(r)$ is shown in the second row of **Figures 3a–c**. The position of these peaks in the model is determined by the average firing rate r_l of losing MCUs in active HCU, the average firing rate r_w of winning MCUs in active HCU, and the average firing rate r_s of MCUs in inactive HCU. The constants r_l , r_w , and r_s are measured directly from BCPNN simulations. Since the firing rate is determined by the soft-WTA process inside a single HCU, the only factor that influences these constants is the number of MCUs in each HCU (M), normally ranging from 10 to 100. Different BCPNN applications might have a different value for these constants. An investigation of both feed-forward and recurrent BCPNN suggests that r_l and r_w are very concentrated and close to 0 and r_{max} , respectively. The value of r_s is analytically determined by assuming all MCUs in inactive HCU have a uniform firing

rate. To represent the general firing rate distribution, we use a set of constants that we obtained from the simulation of the BCPNN as an associative memory via a BCPNN GPU simulator (Herenvarno, 2019). These constants are listed in **Table 1**.

We can now calculate the expectancy of r based on Equation (7) and the constants in **Table 1**. The Equation (8) shows the method to calculate such expectancy. It shows that the expectancy is always very close to r_s regardless of α and M .

$$\begin{aligned} \langle r \rangle &= \sum_{r \in \{r_l, r_w, r_s\}} r \cdot P(r) \\ &= \alpha \left(r_w \frac{1}{M} + r_l \frac{M-1}{M} \right) + (1 - \alpha) r_s \\ &\approx \alpha \left(r_{max} \cdot \frac{1}{M} + 0 \cdot \frac{M-1}{M} \right) + (1 - \alpha) \frac{r_{max}}{M} \\ &= \frac{r_{max}}{M} = r_s \end{aligned} \quad (8)$$

From the recordings of cortical memory systems, it is clear that spikes typically come in the form of bursts (Lundqvist et al., 2016). The BCPNN implementation is optimized for this kind of behavior as its internal representation does not change too frequently with respect to the simulation step. Therefore, when analyzing spike sequences, we assume that the firing probability does not change for the whole spike sequence in the observing period (typically < 200 ms). When at some simulation step, we observe a firing rate r , it is almost certain that at one step earlier, the firing rate was also r .

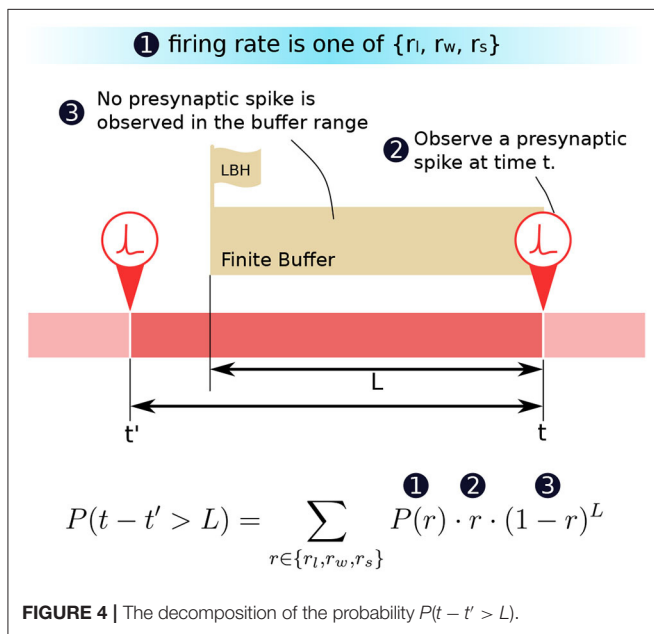
2.4.2. Spike History Buffer

We are now in a position to formulate how the probability $P(t - t' > L)$ can be computed in terms of the firing rate r and its probability distribution $P(r)$. We know that (1) in a period, the probability of the presynaptic neuron firing at a particular firing rate r is $P(r)$. (2) The probability of a presynaptic spike appearing at instance t is then equal to the firing rate r of that period. When

TABLE 1 | Firing rate constants.

<i>M</i>	10	20	30	40	50
r_l	0.0000990	0.0000981	0.0000972	0.0000963	0.0000954
r_w	0.0991090	0.0981361	0.0971812	0.0962443	0.0953254
r_s	0.0100000	0.0050000	0.0033333	0.0025000	0.0020000

<i>M</i>	60	70	80	90	100
r_l	0.0000945	0.0000936	0.0000927	0.0000918	0.0000909
r_w	0.0944245	0.0935416	0.0926767	0.0918298	0.0910009
r_s	0.0016667	0.0014286	0.0012500	0.0011111	0.0010000



there is indeed a presynaptic spike, a row update will trigger the proposed CUE scheme. (3) Given point (2) we can deduce that the probability that no spikes appeared for the previous L ms is $(1 - r)^L$. If t' is the instance when the row was last updated, then the probability of having $(t - t') > L$ is exactly the combination of the three conditions described above. **Figure 4** illustrates the condition and time relation.

Figure 5 shows the plot of such a probability based on the equation in **Figure 4**. The horizontal axis represents the buffer size L . The vertical axis is in logarithmic scale and represents the probability of requiring to look beyond the LBH. When $\alpha = 0$, all MCUs will have a uniformly distributed firing rate r_s . Thus, all the curves in the first sub-figure are straight lines. When $\alpha = 1$, there are two types of MCUs that fire under r_l or r_w , which gives the curve two distinct parts. The first part of the curve is dominated by the effects of winning MCUs, where the probability drops very fast. The second part is almost flat, which is dominated by losing MCUs. The sub-figure in the center with $\alpha = 0.5$ is the combination of the above scenarios. The BCPNN configuration parameter M also affects the shape of the curve.

When M is bigger, the r_s will become closer to 0, making the probability smaller. The overall average firing rate of active HCUs is also roughly equaled to r_s . That is why all three sub-figures, though with different α , have curves starting at the same points, which are determined by r_s .

The most common BCPNN applications have most of their HCUs being active. Thus, the majority of their MCUs are either winning or losing after the soft-WTA process. According to the third sub-figure in **Figure 5**, we choose the $L = 100$ as the size of the spike buffer since it already passes the turning point. Further increasing the buffer size will not dramatically decrease the error probability.

2.4.3. Approximation Function

We have briefly discussed the approximation function and have explained why imperfect approximation function will introduce errors. Although it is impossible to avoid introducing errors with any imperfect approximation function, a well-designed approximation function can introduce small errors in a balanced manner. Therefore, we need to choose a proper tolerance ϵ so that the error bound can reflect the approximation function's predictive ability. In this work, we choose $\epsilon = 1\%$ as the threshold. It is a practical way to consider the error to be much less than the actual value in insensitive systems, such as neural networks. It thus could guarantee that the behavior of the BCPNN remains unaffected. We use ϵ to calculate the error bound of approximation functions.

It is very complex to analytically find out the error propagation from the firing rate r to synaptic weights w_{ij} , as discussed in section 2.1. The calculation of w_{ij} includes many low-pass filters, random number generation, and complex arithmetic operations, such as logarithm. Instead, we opt to find out $P(\text{err} > \epsilon | \mathcal{H})$ by experimental simulation for each type of approximation functions that have been developed. Each experiment has been repeated for more than 10^5 times to guarantee the generality of the collected statistics.

2.4.3.1. Static approximation function

In this subsection, we propose a type of approximation function that we call static. The static approximation function \mathcal{H}_s is defined in Equation (9), where $x \sim \mathcal{U}(0, 1)$ is a uniformly distributed random number generated at each prediction. This function is static because it always uses a constant firing rate to make a prediction. We use the s subscript to indicate that the function is static.

$$\mathcal{H}_s: \mathbb{N}^0 \rightarrow \mathbb{B}, \text{ so that } \mathcal{H}_s: t \mapsto (x < r_s) \quad (9)$$

To maximize the probability for long term correct prediction, the predicted firing rate has to match the true firing rate. However, a spike can be generated according to any of r_l , r_w , and r_s . Therefore, the true firing rate is not constant. We choose r_s to be the static predicted firing rate is because the expectancy of r is $\langle r \rangle = r_s$. It will statistically introduce both positive and negative errors since it does not exactly match the true firing rate.

Figure 6 shows the simulation result when using a static approximation function. We can see that the prediction differs

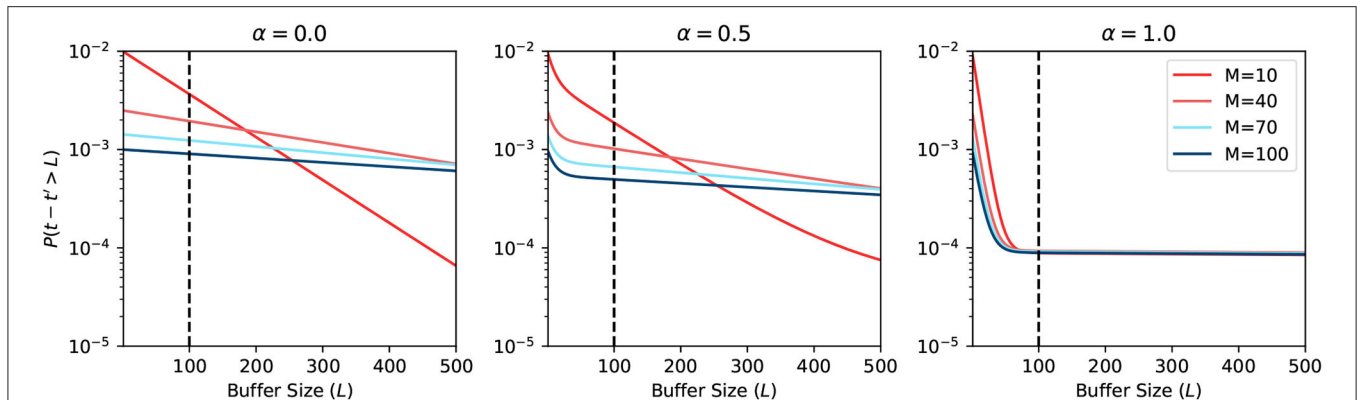


FIGURE 5 | Probability of looking beyond LBH.

when α changes. However, the change is not significant, because the firing rate expectancy roughly equals to r_s , no matter what the value of α is. On the other hand, when the BCPNN configuration changes, the probability changes a lot. This change happens because, for large M , the expected firing rate is close to 0. Since the expected firing rate approaches 0, it makes it easier for the approximation function to have a correct prediction.

2.4.3.2. Adaptive approximation function

The static approximation function \mathcal{H}_s is very simple and unable to properly grasp the three different firing rates in the BCPNN. By modifying it, we can easily implement another type of approximation function that factors in the dynamic information available at evaluation time. We call this type of approximation function as adaptive approximation function. It is defined in Equation (10), where r^* is the nearest recorded firing rate history and $x \sim \mathcal{U}(0, 1)$ is a uniformly distributed random number generated at each prediction. The a subscript in \mathcal{H} indicates that it is an adaptive function. The \mathcal{H}_a is very similar to \mathcal{H}_s . The difference between the two functions is that the predicted firing rate here adapts to the true firing rate r^* .

$$\mathcal{H}_a : \mathbb{N}^0 \times \mathbb{R} \rightarrow \mathbb{B}, \text{ so that, } \mathcal{H}_a : t \mapsto (x < r^*) \quad (10)$$

We record the HCU activation status and winning/losing status of each MCU, assuming that spikes come in bursts. The record of such a status can be reasonably infrequent. Additionally, the required information is boolean and thus occupies very little storage space. If we record the winning/losing status of each MCU every 300 simulation steps, only 32 records are needed to cover nearly 10,000 simulation steps. The cost to record such a range for each MCU is equivalent to just a single integer word. Since we assume the HCU and MCU status will not change frequently, we can use the recorded status to determine the real status for any simulation step.

Figure 7 shows the simulation result when using an adaptive approximation function. We can see that the prediction differs more when α changes, compared to the previous static approximation function curve. The explanation is that the adaptive function adapts to the true firing rate. We note here

again that for large α , the firing rate has a higher probability of being close to 0. Hence it is easier to predict. When the BCPNN configuration changes, the probability changes a lot. The change follows the same trend of static approximation function simulation.

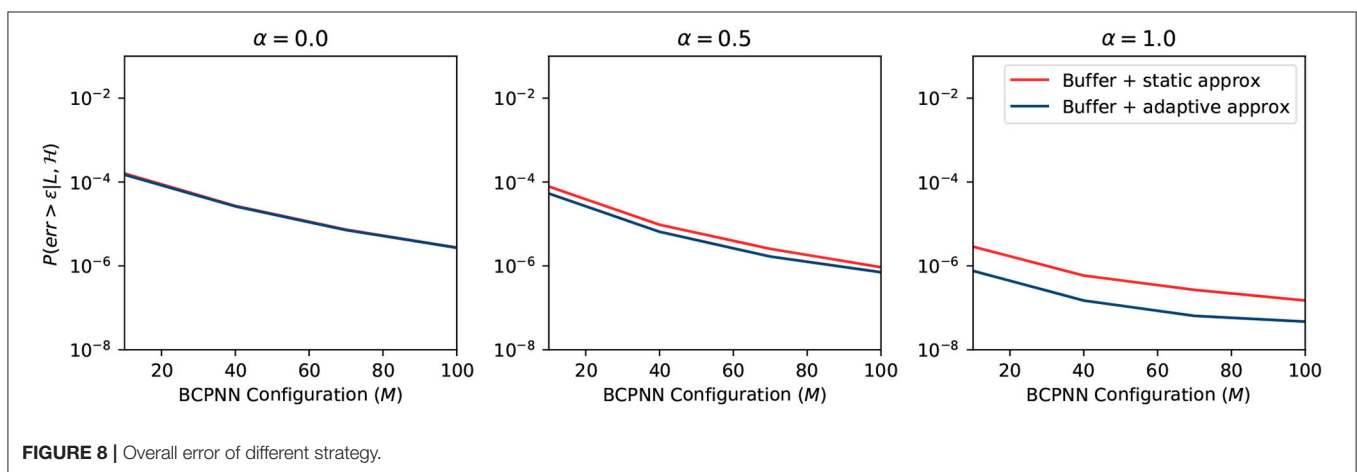
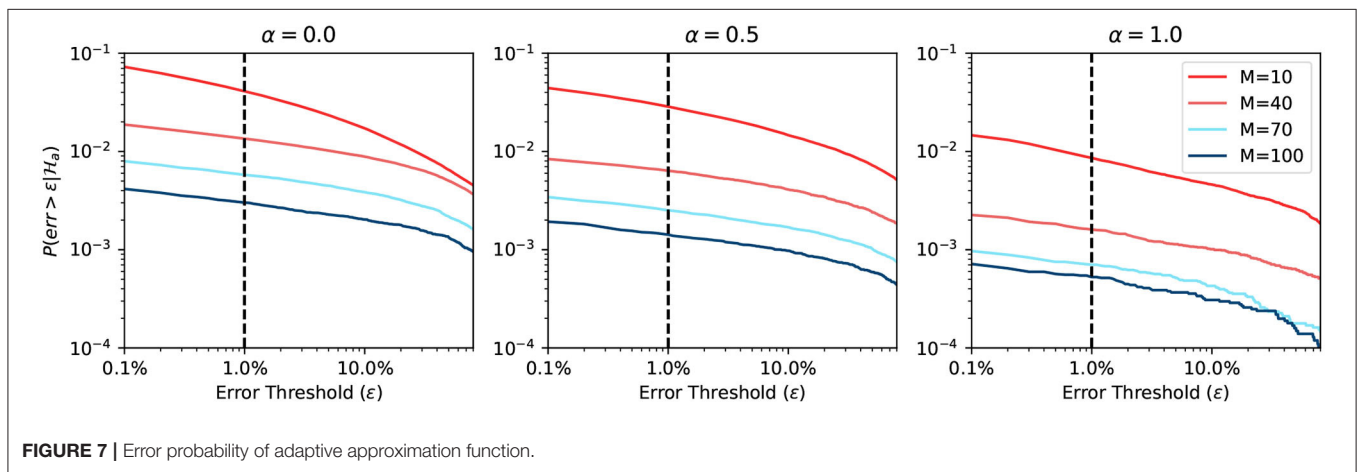
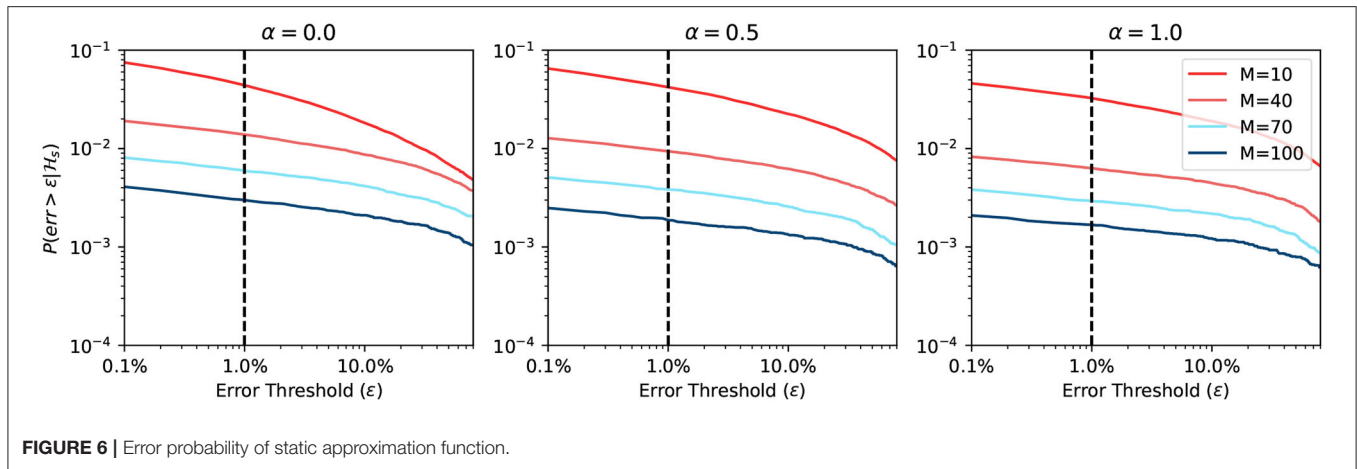
2.4.4. Summary

We plot the overall error probability in **Figure 8**. The curve is the multiplication of previous curves in **Figures 5–7**. It demonstrates two different strategies: (1) buffer + static approximation function and (2) buffer + adaptive approximation function. One can easily tell from the plot that the second strategy dominates the first one. By combining the spike buffer and adaptive approximation functions, we can bound the error as low as to the order of 10^{-4} for any activation level. The worst-case scenario of $\alpha = 0$ should not happen in any BCPNN simulation. Whereas, for normal BCPNN simulations, $\alpha = 1$ is usually guaranteed. The M is commonly set to 100 when simulating a relatively big network. In such cases, the error bound will be improved dramatically to the order of 10^{-8} . Therefore, in normal circumstances, all errors introduced can be considered minor and negligible for the operation of the BCPNN.

The static approximation function leads to a very good error bound as well. Though slightly worse than the adaptive approximation function, it does not require any extra resources. Consequently, it is very efficient to apply the combination of a spike buffer and a static approximation function for most applications. Therefore, we adopt this strategy for the experiments in the results section.

3. EXPERIMENT RESULTS

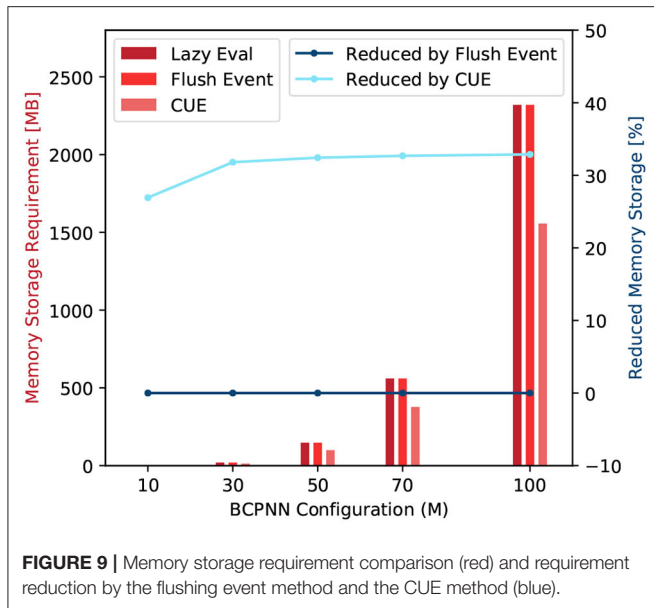
In this section, we set up a series of experiments on a GPU platform to examine all the metrics affected by the CUE method. The GPU we use is an Nvidia Quadro K1200 GPU, which can be profiled by the Nvidia GPU profiling tool—*nvprof*. The profiled metrics include storage and performance aspects. We mainly analyze the storage requirements, memory access demand, and memory access efficiency. The approximation function of CUE used in these



experiments is the static approximation function due to its simplicity, see section 2.4.4. The experiments are designed to compare the original lazy evaluation method (baseline), the flushing event method, and the CUE method. Both the flushing event method and the CUE method have the same buffer size.

3.1. Storage Analysis

In the original lazy evaluation method, we need several variables to be stored in the memory to represent the state of each synapse. This variables are used to track the change of all the traces and time, and they are p_{ij} , e_{ij} , z_{i2} , z_{j2} , w_{ij} , and t_{ij} . Once the column update procedure is removed, all synaptic traces will be updated



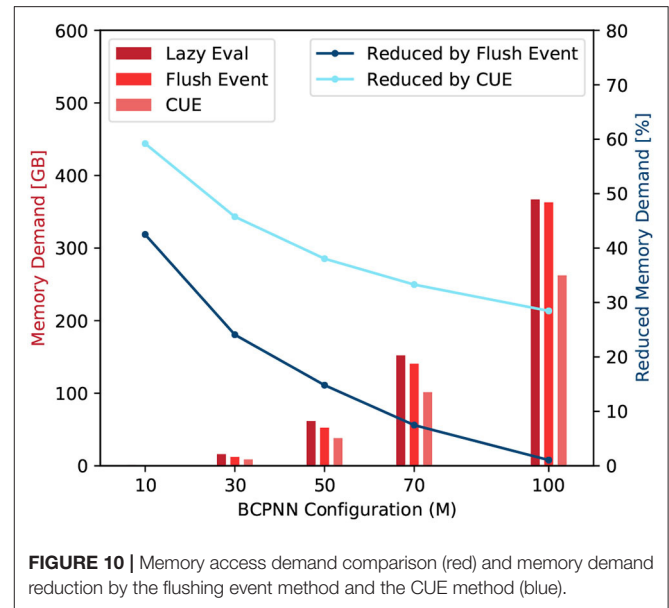
only when a presynaptic spike s_i occurs. Thus, the timestamp t_{ij} will be identical to the timestamp t_i in presynaptic traces and can be safely removed. The z_{i2} is the equivalent z_i trace, and it is stored in the synaptic matrix. In the lazy evaluation method, z_i and z_{i2} could be updated at different times, so we have to make a copy (z_{i2}) inside the synaptic matrix. After applying CUE, z_{i2} will only be updated when an s_i occurs, its value should be identical to z_i for all synapses corresponding to the same presynaptic trace. We can now safely remove the z_{i2} trace.

The flushing event method still keeps the column update. The storage of the flushing event method thus is identical to the lazy evaluation method except that it requires some extra buffer storage for post-synaptic spikes. Overall, the flushing event method doesn't change the storage requirement.

Note that both t_{ij} and z_{i2} traces are synaptic variables that dominate the storage cost for BCPNN simulation because the amount of data representing synaptic traces is proportional to the product of the number of pre- and post-synaptic units. **Figure 9** summarizes the storage comparison of the original lazy evaluation method and CUE method. The figure shows that we save around 33% of memory by applying CUE. The 33% of reduction is due to the elimination of t_{ij} and z_{i2} , which are two out of six synaptic traces.

3.2. Performance Analysis

In this section, we analyze the performance of each method in terms of the memory access demand, the memory access efficiency, and the latency of the row/column update CUDA kernels. The first two factors focus on the memory aspects but neither of them can characterize the performance alone. The overall performance is instead characterized by the latency of the row/column update CUDA kernels. In our experiments, we implement and test a series of small BCPNNs with configuration $M \times M$, where $M \in \{10, 30, 50, 70, 100\}$. Each of these networks is trained to remember 10 patterns, where each pattern is trained for 500 ms.



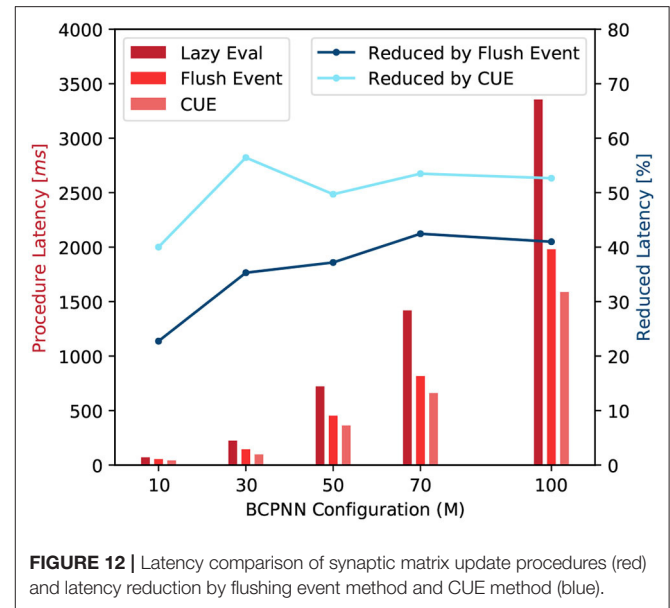
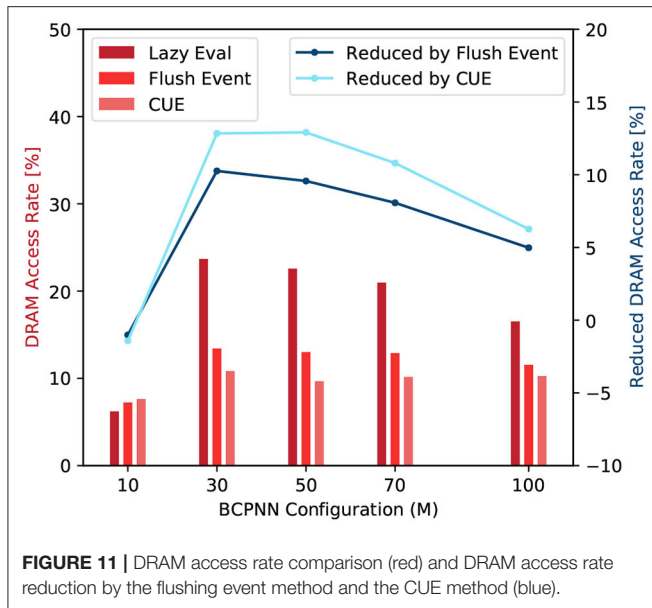
3.2.1. Memory Access Demand

Compared with the original lazy evaluation method, the memory access demand requested by the column update is eliminated. We remind the reader that, in the original lazy evaluation method, the column update roughly demands the same amount of memory access as a row update. However, when applying CUE, the memory access demand of a row update increases slightly due to the access for spike buffer. Therefore, the improvement in memory demand should be $<50\%$.

We record two global memory access related metrics: *global memory load transactions* (N_{ld}) and *global memory store transactions* (N_{st}). They represent the global memory access demand requested by the CUDA kernels. The Equation (11) calculates the memory demand of those row/column update kernels.

$$D = k_{L1} * (N_{ld} + N_{st}) \quad (11)$$

The scaling factor k_{L1} is set to 128 because the size of the Quadro K1200 GPU's L1 cache line is 128 Bytes. The D represents the amount of data in Bytes that is needed to be loaded from, or stored to, the global memory. **Figure 10** shows the comparison among the original lazy evaluation method, the flushing event method, and the CUE method. From the figure, we can see that half of the memory access demand of a small network is eliminated by CUE method due to the elimination of the column update process. In a bigger network, the memory demand is improved by one third. The flushing event method however reduces the memory access demand in a much smaller proportion. It can reduce the invocation of column update, and save memory access for active rows. However, the access of the spike buffer is a non-negligible overhead. Overall, the flushing event method saves memory access demand. However, for big networks, the saving is largely canceled by the overhead of accessing the spike buffer.



3.2.2. Memory Access Efficiency

In this experiment, we profile the following four metrics: *DRAM read transactions* (N_{rd}), *DRAM write transactions* (N_{wr}), *global memory load transactions* (N_{ld}), and *global memory store transactions* (N_{st}). The N_{rd} and N_{wr} are the recorded READ and WRITE operation count from/to the DRAM. They represent the actual DRAM access. The Equation (12) defines a parameter called DRAM access rate γ .

$$\gamma = \frac{k_{L2}(N_{rd} + N_{wr})}{k_{L1}(N_{ld} + N_{st})} \times 100\% \quad (12)$$

Here, the k_{L2} and k_{L1} are the L2 and L1 cache line size, respectively.

From Equation (12), we can see that a large γ value means a large number of real DRAM access and a high cache-miss rate. **Figure 11** shows the DRAM access rate of several BCPNN configurations. We can see that the DRAM access rate is reduced by the CUE method, although the reduction is less dramatic for bigger networks. This is because the column-wise memory access demand has already been eliminated; the improvement has been shown in **Figure 10**. The improvement that is shown in **Figure 11** only presents the average memory access efficiency using the CUE method, compared to the lazy evaluation method.

We can also observe that the flushing event method improves memory access efficiency compared to the lazy evaluation method. It is because even though the memory access demand for the spike buffer is equivalent to the memory access demand of a single cell column update, the memory access for the buffer is coalesced, thus much more efficient than the column update. When the flushing event happens, it also updates according to the spike buffer and resets the spike buffer to reduce the invocation of the column update kernel. However, the flushing event method is not as good as the CUE method because it still keeps the extremely inefficient column update.

If we look at the trend of the DRAM access rate, we can see that it behaves abnormally when $M = 10$. That is because the network is so small, the L2 cache can hold almost the entire network. Therefore, the lazy evaluation method is efficient. For $M \geq 30$, the column update becomes slightly more efficient with the increase of network size. It's mainly due to the variation of the proportion of empty rows in the synaptic matrix that affects the efficiency of the column update. We don't explain the complete causality of this phenomena since it's not very related to our memory access efficiency comparison and it requires a lot of implementation details related to the GPU platform.

3.2.3. Latency

With the improvement in terms of memory demand and memory access efficiency, the overall latency of function call is reduced. We test the average latency of a row and column update procedures. **Figure 12** shows that compared to the original lazy evaluation method, which requires both row and column update, our CUE strategy reduces the latency of the synaptic matrix update. The average latency is reduced by about 50%. The reduction in latency is mainly due to memory optimization and the elimination of issuing the column update kernel, which has some overhead.

The flushing event method also improves the overall performance compared to the lazy evaluation method. However, due to the column update process, the improvement is less than the CUE method which eliminates the entire column update.

For both blue lines in **Figure 12**, when $M = 50$, the reduction drops a little. It is because that the column update efficiency changes with the change of network size. But the change rate is not very uniform due to the nature of the GPU platform. It will be more efficient if the network configuration makes the memory access pattern to fit the GPU warp size. $M = 50$ is one of such configuration points.

4. DISCUSSION

In this paper, we have discussed the BCPNN memory access problem introduced by the lazy evaluation method. An algorithmic optimization has been proposed to tackle this issue. The proposed Column Update Elimination (CUE) method eliminates the column update and merges it with the row update, with the help of spike history buffer and approximation function. Using the CUE method, we gain not only memory access efficiency but also other improvements, such as the reduction of memory storage, memory access demand, etc. We also show that our algorithmic modification only introduces negligible errors and does not compromise the functionality of the BCPNN.

In this section, we further examine the potential of the proposed method. We focus on the new possibilities after the column update has been eliminated, and other learning rules that CUE method can fit in. Finally, based on what we have achieved, we describe an outlook that could improve BCPNN simulation even further.

4.1. Exploiting the Temporal Locality of Spike Train

Memory access efficiency can be further improved by architectural optimization. We have analyzed the pattern of spike train in section 2.4.1. Spikes are generated in burst mode regulated by stimulus pattern. Usually, the change of these stimulus patterns is infrequent. Thus, when the neural network is in the middle of a stable stimulus pattern, the firing patterns of spikes are also stable.

We have analyzed the percentage of winning MCUs in both active and silent HCUs. Even in completely active $H \times M$ BCPNN ($\alpha = 1$), the fraction of winning MCUs which fire frequently is just $\frac{1}{M}$. Therefore, only a fraction of $\frac{1}{M}$ connections will be active at each simulation step, assuming uniform interconnections. Therefore, when the firing patterns are stable, the part of synaptic traces that need to be frequently updated is also stable, and the global memory access patterns of synaptic traces are stable as well.

In this paper, we have eliminated the non-coalesced column-wise memory access pattern. With stable global memory access patterns, we can cache the frequently updated fraction of synaptic traces by designing a large enough cache between the memory and computation unit. The single global memory access pattern guarantees that the cache system will not be interrupted by other memory access patterns.

Unfortunately, the estimated size of such a cache usually is much bigger than any off-the-shelf commercial computer architecture. The insufficient size of the cache will lead to frequent swapping data between the cache and DRAM, thus significantly compromises the memory access efficiency. The customization of the cache system becomes necessary and can only be done in custom hardware architecture, such as ASICs.

4.2. CUE Method for STDP

The STDP learning rule changes the synaptic weights based on the correlation of pre- and post-synaptic spikes. The amount

of change depends on the time difference between the pre- and post-synaptic spikes. A pre- and post-synaptic spike pair for a synaptic connection $\langle s_i, s_j \rangle$ occurs at time $\langle t_i, t_j \rangle$. If $t_i < t_j$, they are correlated. Otherwise, they are anti-related. One commonly used method to calculate such causal strength is described in Equation (13).

$$\Delta w_{ij} = \begin{cases} +A_+ \cdot e^{-\frac{t_j - t_i}{\tau}}, & \text{if } t_i < t_j \\ -A_- \cdot e^{-\frac{t_i - t_j}{\tau}}, & \text{if } t_i > t_j \end{cases} \quad (13)$$

We can see that the update of weights in the STDP learning rule is triggered by pre- and post-synaptic spikes, just like the BCPNN learning rule. It is natural to organize the synaptic weights of STDP learning as a matrix stored in the memory. Because of its triggering mechanism and data structure, the STDP learning rule will suffer from the same dual memory access pattern issue as a lazy evaluation method.

By proposing the same solution to STDP learning, we can use a post-synaptic buffer and an approximation function to delay the update of the post-synaptic triggered column update. The buffer size L , and the type of approximation function \mathcal{H} will be different for STDP learning. Further analysis and simulation should be done to investigate the error bound. For example, the probability density function (PDF) of the firing rate in STDP might be different. It might be enough for STDP by just using the spike history buffer. However, the principle of optimizing memory access efficiency remains the same for both STDP and BCPNN learning rules.

4.3. Outlook

In the future, we could also explore the option of hardware architecture to improve the BCPNN simulation further. For example, we could dimension a big enough cache that can hold the complete stationary synaptic traces to avoid DRAM memory access. We could also use non-von Neuman architecture, such as memristor, which could potentially avoid the memory access problem. Other algorithmic modifications combined with approximate computing, such as delayed stochastic row update, could also improve the overall BCPNN simulation.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

The initial idea proposed in the paper came from AL and AH. YY further developed the method by introducing the approximation function, proposed the methodology for error analysis, and performed the experiments to verify the proposed method. DS and RJ helped with the refinement of method especially the error analysis part. All authors contributed to the article and approved the submitted version.

REFERENCES

- Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J. P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Netw.* 32, 339–348. doi: 10.1016/j.neunet.2012.02.022
- Buxhoeveden, D. P., and Casanova, M. F. (2002). The minicolumn hypothesis in neuroscience. *Brain* 125, 935–951. doi: 10.1093/brain/awf110
- Coultrip, R., Granger, R., and Lynch, G. (1992). A cortical model of winner-take-all competition via lateral inhibition. *Neural Netw.* 5, 47–54. doi: 10.1016/S0893-6080(05)80006-1
- Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359
- Davies, S., Galluppi, F., Rast, A. D., and Furber, S. B. (2012). A forecast-based STDP rule suitable for neuromorphic implementation. *Neural Netw.* 32, 3–14. doi: 10.1016/j.neunet.2012.02.018
- Dayan, P., and Abbott, L. F. (2001). *Neural Encoding I: Firing Rates and Spike Statistics, Chapter 1*. Cambridge, MA: MIT Press.
- Farahini, N., Hemani, A., Lansner, A., Clermidy, F., and Svensson, C. (2014). “A scalable custom simulation machine for the Bayesian confidence propagation neural network model of the brain,” in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (Singapore), 578–585. doi: 10.1109/ASP-DAC.2014.6742953
- Fiebig, F., Herman, P., and Lansner, A. (2020). An indexing theory for working memory based on fast hebbian plasticity. *eNeuro* 7:ENEURO.0374-19.2020. doi: 10.1523/ENEURO.0374-19.2020
- Griffiths, D. F., and Higham, D. J. (2010). *Euler's Method, Chapter 2*. London: Springer.
- Herenvarno (2019). *GSBN: GPU Version of Spiking-Based BCPNN*. Available online at: <https://github.com/herenvarno/gsbnn>
- Hubel, D. H., and Wiesel, T. N. (1974). Uniformity of monkey striate cortex: a parallel relationship between field size, scatter, and magnification factor. *J. Compar. Neurol.* 158, 295–305. doi: 10.1002/cne.901580305
- Jin, X., Rast, A., Galluppi, F., Davies, S., and Furber, S. (2010). “Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware,” in *Proceedings of the International Joint Conference on Neural Networks*, (Barcelona), 1–8. doi: 10.1109/IJCNN.2010.5596372
- Johansson, C., and Lansner, A. (2007). Towards cortex sized artificial neural systems. *Neural Netw.* 20, 48–61. doi: 10.1016/j.neunet.2006.05.029
- Knight, J. C., and Furber, S. B. (2016). Synapse-centric mapping of cortical models to the spinnaker neuromorphic architecture. *Front. Neurosci.* 10:420. doi: 10.3389/fnins.2016.00420
- Knight, J. C., Tully, P. J., Kaplan, B. A., Lansner, A., and Furber, S. B. (2016). Large-scale simulations of plastic neural networks on neuromorphic hardware. *Front. Neuroanat.* 10:37. doi: 10.3389/fnana.2016.00037
- Lansner, A., and Ekeberg, Ö. (1989). A one-layer feedback artificial neural network with a bayesian learning rule. *Int. J. Neural Syst.* 1, 77–87. doi: 10.1142/S0129065789000499
- Lansner, A., and Holst, A. (1996). A higher order Bayesian neural network with spiking units. *Int. J. Neural Syst.* 7, 115–128. doi: 10.1142/S0129065796000816
- Lennie, P. (2003). The cost of cortical computation. *Curr. Biol.* 13, 493–497. doi: 10.1016/S0960-9822(03)00135-0
- Li, C., Yang, Y., Feng, M., Chakradhar, S., and Zhou, H. (2016). “Optimizing memory efficiency for deep convolutional neural networks on GPUs,” in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC* (Salt Lake City, UT: IEEE Computer Society), 633–644. doi: 10.1109/SC.2016.53
- Lundqvist, M., Rehn, M., Djurfeldt, M., and Lansner, A. (2006). Attractor dynamics in a modular network model of neocortex. *Network: Computation in Neural Systems* 17, 253–276. doi: 10.1080/09548980600774619
- Lundqvist, M., Rose, J., Herman, P., Brincat, S. L., Buschman, T. J., and Miller, E. K. (2016). Gamma and beta bursts underlie working memory. *Neuron* 90, 152–164. doi: 10.1016/j.neuron.2016.02.028
- Markram, H., Gerstner, W., and Sjöström, P. J. (2012). Spike-timing-dependent plasticity: a comprehensive overview. *Front. Synap. Neurosci.* 4:2. doi: 10.3389/978-2-88919-043-0
- Meli, C., and Lansner, A. (2013). A modular attractor associative memory with patchy connectivity and weight pruning. *Netw. Comput. Neural Syst.* 24, 129–150. doi: 10.3109/0954898X.2013.859323
- Morrison, A., Aertsen, A., and Diesmann, M. (2007). Spike-timing-dependent plasticity in balanced random networks. *Neural Comput.* 19, 1437–1467. doi: 10.1162/neco.2007.19.6.1437
- Mutlu, O. (2013). “Memory scaling: a systems architecture perspective,” in *2013 5th IEEE International Memory Workshop, IMW 2013* (Monterey, CA), 21–25. doi: 10.1109/IMW.2013.6582088
- Pedroni, B. U., Joshi, S., Deiss, S. R., Sheik, S., Detorakis, G., Paul, S., et al. (2019). Memory-efficient synaptic connectivity for spike-timing-dependent plasticity. *Front. Neurosci.* 13:357. doi: 10.3389/fnins.2019.00357
- Prezioso, M., Mahmoodi, M. R., Bayat, F. M., Nili, H., Kim, H., Vincent, A., et al. (2018). Spike-timing-dependent plasticity learning of coincidence detection with passively integrated memristive circuits. *Nat. Commun.* 9, 1–8. doi: 10.1038/s41467-018-07757-y
- Ravichandran, N. B., Lansner, A., and Herman, P. (2020). Brain-like approaches to unsupervised learning of hidden representations—a comparative study. arXiv[Preprint].arXiv:2005.03476.
- Serrano-Gotarredona, T., Masquelier, T., Prodromakis, T., Indiveri, G., and Linares-Barranco, B. (2013). STDP and STDP variations with memristors for spiking neuromorphic learning systems. *Front. Neurosci.* 7:2. doi: 10.3389/fnins.2013.00002
- Sheik, S., Paul, S., Augustine, C., and Cauwenberghs, G. (2016). “Membrane-dependent neuromorphic learning rule for unsupervised spike pattern detection,” in *Proceedings—2016 IEEE Biomedical Circuits and Systems Conference, BioCAS 2016* (Shanghai: Institute of Electrical and Electronics Engineers Inc.), 164–167. doi: 10.1109/BioCAS.2016.7833757
- Stathis, D., Sudarshan, C., Yang, Y., Jung, M., Jafri, S. A. M. H., Weis, C., et al. (2020). eBrainII: a 3 kW realtime custom 3D DRAM integrated ASIC implementation of a biologically plausible model of a human scale cortex. *J. Signal Process. Syst.* 2020, 1–21. doi: 10.1007/s11265-020-01562-x
- Tully, P. J., Lindén, H., Hennig, M. H., and Lansner, A. (2016). Spike-based bayesian-hebbian learning of temporal sequences. *PLoS Comput. Biol.* 12:e1004954. doi: 10.1371/journal.pcbi.1004954
- Vogginger, B., Schüffny, R., Lansner, A., Cederström, L., Partzsch, J., and Höppner, S. (2015). Reducing the computational footprint for real-time BCPNN learning. *Front. Neurosci.* 9:2. doi: 10.3389/fnins.2015.00002
- Yang, Y., Jafri, S. M. A. H., Hemani, A., and Stathis, D. (2017). “MTP-caffe: memory, timing, and power aware tool for mapping CNNs to GPUs,” in *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM '17* (New York, NY: Association for Computing Machinery), 31–36. doi: 10.1145/3029580.3029585
- Yousefzadeh, A., Masquelier, T., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017). “Hardware implementation of convolutional STDP for on-line visual feature learning,” in *Proceedings—IEEE International Symposium on Circuits and Systems* (Baltimore, MD: Institute of Electrical and Electronics Engineers Inc.), 1–4. doi: 10.1109/ISCAS.2017.8050870

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 Yang, Stathis, Jordão, Hemani and Lansner. 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 Computing Using NAND Flash Memory Architecture With Pulse Width Modulation Scheme

Sung-Tae Lee and Jong-Ho Lee*

Department of Electrical and Computer Engineering, ISRC, Seoul National University, Seoul, South Korea

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Alex James,
Independent researcher, Bangalore,
India
Jiyong Woo,
Electronics and Telecommunications
Research Institute (ETRI), South Korea

*Correspondence:

Jong-Ho Lee
jhl@snu.ac.kr

Specialty section:

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

Received: 10 June 2020

Accepted: 17 August 2020

Published: 18 September 2020

Citation:

Lee S-T and Lee J-H (2020)
Neuromorphic Computing Using
NAND Flash Memory Architecture
With Pulse Width Modulation
Scheme. *Front. Neurosci.* 14:571292.
doi: 10.3389/fnins.2020.571292

A novel operation scheme is proposed for high-density and highly robust neuromorphic computing based on NAND flash memory architecture. Analog input is represented with time-encoded input pulse by pulse width modulation (PWM) circuit, and 4-bit synaptic weight is represented with adjustable conductance of NAND cells. Pulse width modulation scheme for analog input value and proposed operation scheme is suitably applicable to the conventional NAND flash architecture to implement a neuromorphic system without additional change of memory architecture. Saturated current-voltage characteristic of NAND cells eliminates the effect of serial resistance of adjacent cells where a pass bias is applied in a synaptic string and IR drop of metal wire resistance. Multiply-accumulate (MAC) operation of 4-bit weight and width-modulated input can be performed in a single input step without additional logic operation. Furthermore, the effect of quantization training (QT) on the classification accuracy is investigated compared with post-training quantization (PTQ) with 4-bit weight. Lastly, a sufficiently low current variance of NAND cells obtained by the read-verify-write (RVW) scheme achieves satisfying accuracies of 98.14 and 89.6% for the MNIST and CIFAR10 images, respectively.

Keywords: neuromorphic, synaptic device, in-memory computing, NAND flash, deep neural networks, quantized neural networks

INTRODUCTION

Recently, deep neural networks (DNNs) have achieved excellent performance for a variety of intelligent tasks, such as natural language processing, computer vision, and speech recognition (Truong et al., 2016; Nishani and Cico, 2017; Sainath et al., 2017). However, recent high-performance DNNs require a vast network size and an enormous number of parameters and computational capability, which demand very fast and power-hungry graphics processing units (Scardapane et al., 2017; Khan et al., 2019). Furthermore, von Neumann architecture leads to tremendous time and energy consumption due to the bottleneck between memory and processor. To accelerate neural network computation, neuromorphic systems that can efficiently process multiply-accumulate (MAC) operation have been proposed and developed utilizing memory devices (Suri et al., 2011; Jackson et al., 2013).

In prior research, resistive random access memories (RRAMs) were mainly used as synaptic devices to implement the neuromorphic system (Park et al., 2013; Tang et al., 2017; Andri et al., 2018; Zhou et al., 2018; Guan and Ohsawa, 2019). However, RRAMs require further research in terms of cell characteristics variation, reliability, and integration of selectors for large-scale integration (Woo and Yu, 2019). In addition, the effect of metal wire resistance can cause inaccurate vector-matrix multiplication (VMM) operation in a large array (Wang et al., 2020). Furthermore, low on/off current ratio of RRAMs restricts bandwidth to sum current of many RRAM devices (Sun et al., 2018; Yu et al., 2020). The state-of-the-art algorithms typically demand a huge parameter size. To satisfy this demand, NAND flash memory can be a promising candidate for a synaptic device to meet this requirement. NAND flash memory offers ultra-high bit density for immense data storage and low fabrication cost per bit, and it has been well known as a mature technology (Yamashita et al., 2017; Kang et al., 2019; Huh et al., 2020). However, NAND flash memory was not commonly used in neuromorphic system because of the characteristics of the string structure. In RRAM crossbar array, the input bias is applied to word-lines (WLs), and output current is summed through bit-lines (BLs). Therefore, VMM of the input voltage applied to the WLs and the conductance of the RRAM can be easily implemented. However, in NAND flash memory architecture, the WL and source-line (SL) are shared by NAND strings in the same block. Furthermore, read bias and pass bias are applied to the selected layer and unselected layers, respectively, to read the current of NAND cells of a selected layer. Therefore, it has been considered difficult to implement VMM in NAND flash memory architecture.

In this article, a novel neuromorphic architecture is proposed for the quantized neural network (QNN) utilizing NAND flash memory with a pulse width modulation (PWM) scheme. Our scheme implements a high-density neuromorphic system because two NAND cells having eight current levels (3-bit) are used as one synaptic device, and a PWM circuitry can represent the analog input values. Furthermore, our scheme can process MAC of the analog input value and 4-bit weight with only a single input step, which considerably decreases power consumption and burden of peripheral circuits needed in architectures in digital design. Utilizing saturated current-voltage characteristics of NAND cells solves the problem arising from the resistance of the pass cells where a pass bias is applied and metal wire. Furthermore, the effect of quantization training (QT) on inference accuracy is investigated compared with post-training quantization (PTQ). Lastly, we show that sufficiently low current variance of synaptic devices obtained by the read-verify-write (RVW) method achieves satisfying accuracy.

MATERIALS AND METHODS

Neuromorphic System Using NAND Flash

Figure 1 shows schematically an operation scheme of a neuromorphic system utilizing a three-dimensional (3D) NAND

flash with PWM circuits. Input voltages with adjustable pulse width from PWM circuits are imposed on string-select lines (SSL), where cell current is added in the BLs, as shown in **Figure 1A**. The NAND cells in the k^{th} WL represent the synapses in the k^{th} synaptic layer of the neural network shown in **Figure 1B**. The read bias (V_{read}) and pass bias (V_{PASS}) are imposed on a selected WL and unselected WLs, respectively, as shown in **Figure 1C**. When V_{read} is imposed on the WL sequentially along the synaptic string, the output of each postsynaptic neuron is sequentially generated. Cells are connected to a selected WL store weights, and each weight determines the string-current of each string. In the proposed scheme, the input voltage is simultaneously imposed on all SSLs. The proposed operation scheme is different from that of the conventional NAND flash memory architecture, as compared in **Table 1**. The input bias corresponding to neuron activation is applied to SSLs, and the current sum is read through BLs in the proposed operation scheme. On the other hand, the cell selected by the input address is read through BL in the conventional NAND flash memory. Furthermore, SSLs are simultaneously biased by input voltage in the proposed scheme, whereas read bias is imposed sequentially on each SSL in the conventional NAND flash memory. Therefore, this scheme significantly reduces latency compared with conventional NAND flash memory technology. The output current is read through the BL in both schemes. In addition, the proposed synaptic architecture utilizing NAND flash is different from the RRAM crossbar array. In the RRAM crossbar array, the input bias is applied to WLs, and the output current is summed through BLs. The NAND cell array is composed of cell strings, and each cell string has multiple cells connected in series. In the NAND cell array, the WL and SL are shared by NAND strings in the same block of NAND flash memory. In addition, to turn on unselected cells, pass bias (V_{PASS}) should be applied to WLs of unselected cells. Therefore, in the proposed synaptic architecture, the input is applied to SSLs, and the output current is read in the BLs. Furthermore, cells in the k^{th} layer in NAND flash strings represent synapses in the k^{th} layer synapse layer in neural networks. Note that the proposed operation scheme can be applied to both 2D and 3D NAND flash memory architectures.

Figure 2 represents VMM operation utilizing a string array and neuron circuits. In the neuromorphic system, the weight and input in the DNN algorithm are represented by conductance and input voltage of synaptic devices, respectively. In the DNN algorithm, weighted sum output is linearly increased with input as shown in the equation;

$$O = \sum WX \quad (1)$$

where O , W , and X represent weighted sum output, weight, and input, respectively. In the neuromorphic system, it is commonly assumed synaptic devices have linear current (I) versus voltage (V) characteristics (Kim T. et al., 2017). If synaptic devices have linear I - V characteristics, the amplitude of input in a DNN model can be simply represented by the amplitude of input voltage of synaptic devices. Then, the weighted sum current is represented by the product of input voltage and

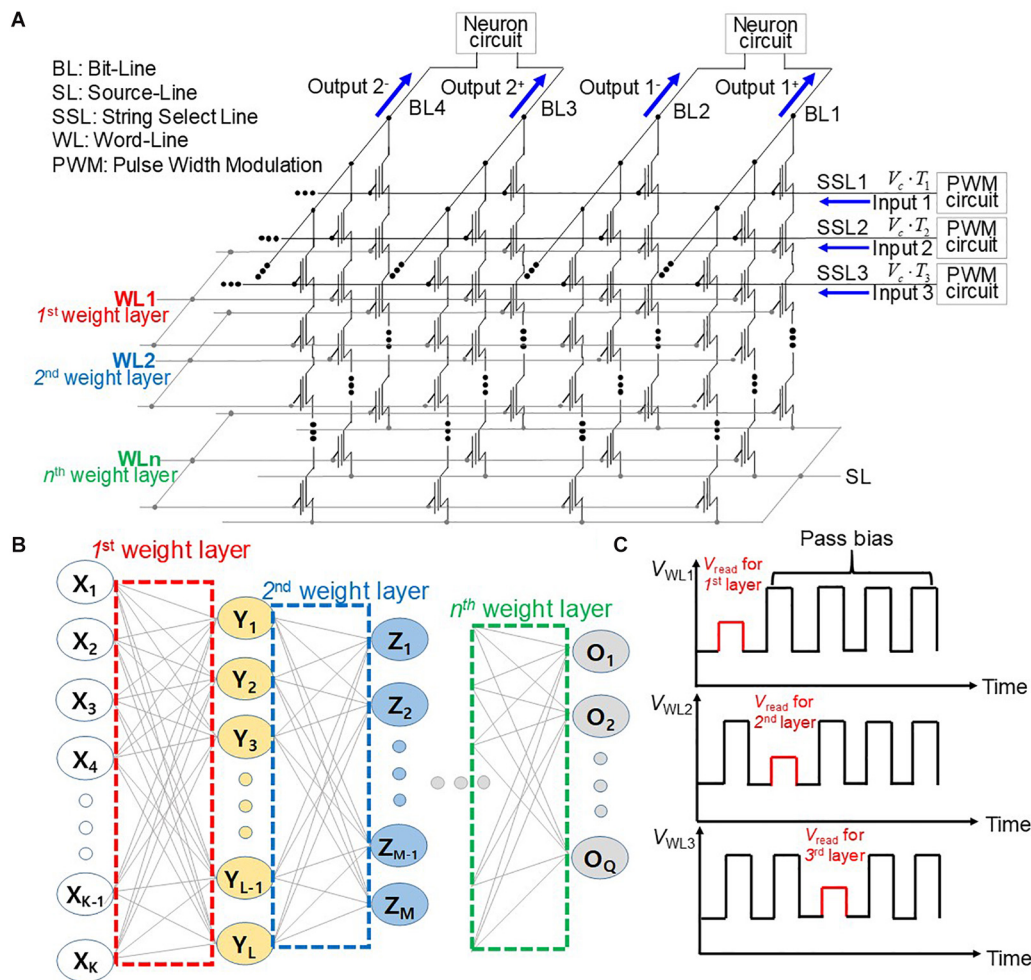


FIGURE 1 | (A) Operation scheme for synaptic string array utilizing NAND flash memory with PWM circuits. **(B)** Schematic diagram of neural networks. **(C)** Pulse diagram applied to WLs with the time.

conductance of synaptic devices, as shown in the equation;

$$I = \sum GV \quad (2)$$

where I , G , and V represent weighted sum current, conductance, and input voltage of devices, respectively. On the other hand, the cell device of NAND flash memory has non-linear I - V characteristics (Lee et al., 2018, 2019a), which means output current has a non-linear relationship with the input voltage.

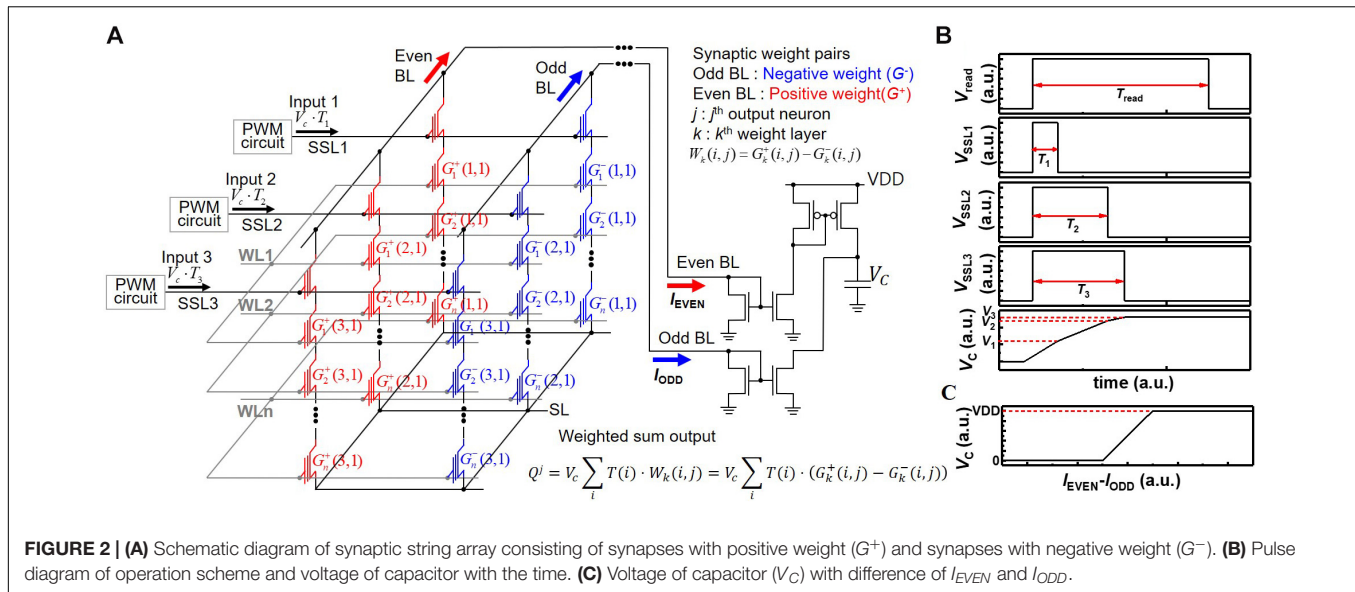
Thus, an analog amplitude of input pulse cannot represent the amplitude of input in a DNN algorithm (Lee et al., 2019b). To resolve the problem of the non-linear I - V characteristic of NAND cells, the PWM scheme is proposed. In this scheme, the amplitude of the input pulse is fixed, whereas the pulse width of the input pulse varies in proportional to the amplitude of input in a DNN algorithm. Then, the weighted sum output is represented by the amount of charge accumulated in neuron circuits, whereas the input voltage is applied as shown in the equation;

$$Q = V \sum GT \quad (3)$$

where Q , V , G , and T represent weighted sum charge, the constant amplitude of input pulse, conductance of device, and pulse width of the input pulse, respectively. Therefore, the weighted sum in a DNN model can be correctly performed in neuromorphic systems by using the PWM scheme despite the non-linear I - V characteristics of cell devices. In addition, this scheme is well fitted to conventional NAND flash memory architecture. Two adjacent NAND cells are used for one synaptic device to

TABLE 1 | Comparison of proposed operation scheme with that of conventional NAND flash memory.

	Proposed operation scheme	Conventional NAND flash memory
Input	Input bias corresponding to neuron activation	Address of a selected cell
String select line	Simultaneously biased	Sequentially read
Output current	Bit-line	Bit-line



represent negative weight value. Considering the negative weight, the charge accumulated in the neuron circuit can be represented by the equation;

$$Q = V \sum T(G^+ - G^-) \quad (4)$$

where G^- and G^+ represent negative and positive weights, respectively.

By adopting two current mirrors and one capacitor as one neuron circuit shown in **Figure 2A**, current summing in time scale and subtracting between positive and negative weights are carried out (Kim H. et al., 2017). In **Figure 2A**, synaptic devices connected to even BL and odd BL have positive weight (G^+) and negative weight (G^-), respectively. The k , j , and i in the weighted sum equation of **Figure 2A** represent the k^{th} synapse layer, j^{th} postsynaptic neuron, and i^{th} synapse connected to j^{th} neuron, respectively. The current of even BL (I_{EVEN}) accumulates the charge in a capacitor, and the current of odd BL (I_{ODD}) reduces the charge in a capacitor. **Figure 2B** represents the pulse diagram of the operation scheme and voltage of the capacitor (V_c) in the case of positive weight as an example. Whereas V_{read} is applied to selected WL during T_{read} , the V_{SSL1} , V_{SSL2} , and V_{SSL3} are applied to SSL1, SSL2, and SSL3 during T_1 , T_2 , and T_3 , respectively. Then, I_1 , I_2 , and I_3 flow through NAND strings 1, 2, and 3, respectively. V_c increases to V_3 , which equals to $(I_1 \cdot T_1 + I_2 \cdot T_2 + I_3 \cdot T_3)/C$. Here, for simplicity of description, it is assumed that the weights of cells to which read bias is applied are the same. The VDD and ground limit the voltage of the capacitor. Therefore, the relationship between V_c and the difference of I_{EVEN} and I_{ODD} represents a hard-sigmoid function, which is one of the activation functions, as shown in **Figure 2C**. Note that V_c linearly increases with the difference of I_{EVEN} and I_{ODD} in a specific current region where the difference of I_{EVEN} and I_{ODD} ranges from $-(C \cdot VDD)/(2 \cdot T_{\text{read}})$ to $(C \cdot VDD)/(2 \cdot T_{\text{read}})$. Here, for simplicity of description, it is assumed that I_{EVEN} and I_{ODD} are constant during T_{read} . Therefore, this scheme can process MAC

of 4-bit weight and analog input pulse and implement neuron activation in a single input step without any logic operation, significantly reducing the burden of peripheral circuits required for logic operation. The PWM circuits, current mirrors, and capacitors are reused for all synapse layers (equivalently WLs) in a synaptic string, which greatly reduces the area of peripheral circuits. Note that the convolution operation and VMM in multilayer neural networks are the same operations in principle when a 2D convolution kernel is unrolled into a 1D column (Gao et al., 2016). Therefore, the proposed scheme in this work can be applied to the implementation of convolutional neural networks.

RESULTS

Measurement Results of NAND Flash Cells

We measured floating-gate 2D NAND cells fabricated with 26-nm technology. One cell string is composed of 64 cells, including a ground select line transistor, an SSL transistor, and two dummy cells. The channel width and length are 20 and 26 nm, respectively. **Figure 3** represents BL current (I_{BL}) versus BL voltage (V_{BL}) curves with various weight levels at a V_{PASS} of 6 V and WL voltage (V_{WL}) of 0 V. Each cell has eight weight levels giving eight current levels from 0 to 1.4 μA , and the current difference between adjacent current levels is 200 nA. As one synaptic device consists of positive and negative weight cells, the synaptic device has a 4-bit weight. In the neuromorphic system, the IR drop of metal wire causes inaccurate VMM operation, as resistance in metal wire decreases effective voltage imposed on synaptic devices. In addition, the channel resistance of adjacent cells where pass bias is applied also results in inaccurate VMM operation in NAND flash memory. To resolve these problems, NAND cells are operated in the saturation region, eliminating the problem caused by the resistance of the metal wire and the pass cells in the unselected layers. I_{BL} rarely changes despite the

change of V_{BL} in the saturation region, as shown in **Figure 3**, and the minimum output resistance of a NAND cell, which operates at a saturation region, is about 20 M Ω .

As V_{PASS} is applied to pass cells during the inference process, V_{PASS} disturbance needs to be investigated. **Figure 4** shows the I_{BL} - V_{WL} curves with V_{PASS} disturbance and 12-V program bias (V_{PGM}). Black square symbols represent the I_{BL} - V_{BL} curve measured in a fresh cell. The red circle symbol represents the I_{BL} - V_{BL} curve after applying V_{PASS} of 6 V 10^4 times to the fresh cell. As these two curves are nearly the same, the effect of V_{PASS} is negligible. The curves measured after a pulse with V_{PGM} of 12 V is applied to the cell 10 times and 20 times, which are depicted by green triangle symbols and blue diamond symbols, respectively. The inset shows the change of I_{BL} (ΔI_{BL}) after applying 10^4 V_{PASS}

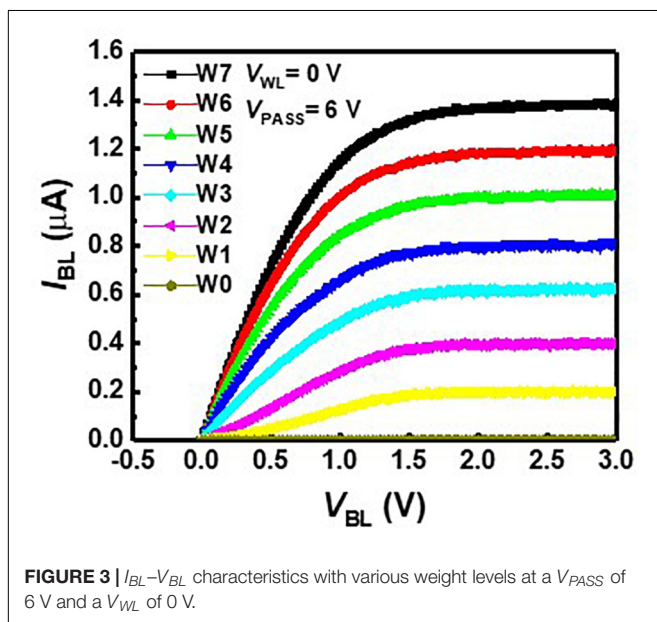


FIGURE 3 | I_{BL} - V_{BL} characteristics with various weight levels at a V_{PASS} of 6 V and a V_{WL} of 0 V.

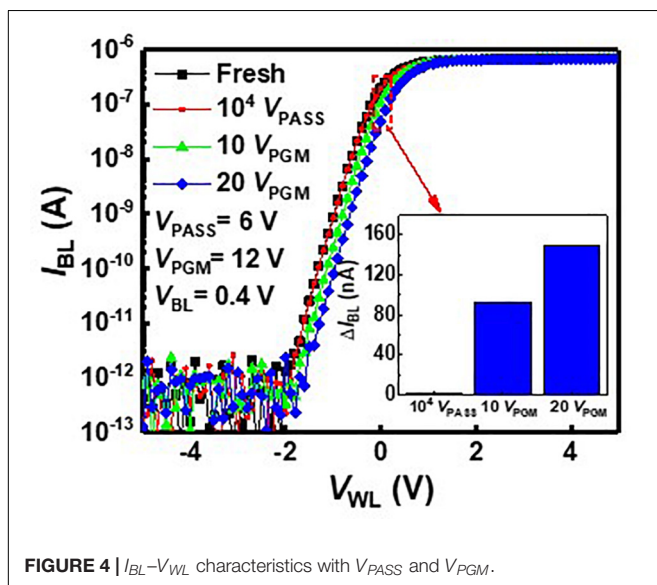


FIGURE 4 | I_{BL} - V_{WL} characteristics with V_{PASS} and V_{PGM} .

(6 V), 10 V_{PGM} , and 20 V_{PGM} pulses. As shown in the inset, the I_{BL} shows little variation with 10^4 V_{PASS} pulses compared with 10 V_{PGM} pulses.

We estimate device variation, as it degrades the classification accuracy of neural networks. RVW method is used to match I_{BL} of NAND cells in a NAND array to the target current level among eight levels in **Figure 3**. The weights obtained in off-chip training are transferred to cells by the RVW method, which reiterates the cycle of reading, verifying, and writing threshold voltage of NAND cells. After each V_{PGM} pulse is imposed on the NAND cell, the I_{BL} of the NAND cell is measured by the V_{read} to check if the measured conductance of the cell is outside of the target conductance range. A V_{PGM} is imposed on the NAND cell if the conductance is outside of the target conductance range. As this process is repeated, the amplitude of V_{PGM} increases. The RVW process ends when the conductance of the cell is within the target conductance range. In this work, ~40 pulses are applied to fit the current of a synaptic device within the range of target current on average, and amplitude of V_{PGM} increases from 11 V with a fixed width of 100 μ s. **Figure 5** shows the measured I_{BL} distribution of second and third weight levels (W2, W3) obtained by the RVW method in the NAND string. To investigate the effect of device variation on neural networks, the largest variation among the eight levels need to be estimated. Among the eight levels, W2 has the largest device variation, and W3 has the smallest device variation. The estimated device variation (σ_w/μ_w) of W2 is 3.43%, and W3 is 1.68% based on the statistical parameters extracted from the measurement data. In this estimation, we assume that the conductance distribution of NAND cells follows a Gaussian distribution (Lee et al., 2019b).

Pulse Width Modulation Circuit

Figure 6 represents a PWM circuit consisting of a sawtooth generator, a differential amplifier, and a level shifter. The sawtooth generator produces a sawtooth wave (V_S). The differential amplifier compares V_S with an analog signal (V_A) and amplifies the difference between V_S and V_A . The level shifter produces a width-modulated pulse (V_P) with a fixed amplitude, and V_P is applied to SSLs of a synaptic string array. **Figure 7** shows the simulation results of V_A , V_S , and V_P in the PWM circuit when V_A s are 0.3 and 0.9 V, as an example. The pulse width of the V_P is proportional to the amplitude of the V_A . As the amplitude of V_A increases from 0.3 to 0.9 V, the pulse width of V_P increases from 3 to 9 μ s.

Evaluation of Quantized Neural Networks

In QNNs, the weight can be quantized during or after training. PTQ means that training the DNNs with high-precision floating-point weight without quantization during training. After the training process, PTQ quantizes the pretrained weight at the inference stage. On the other hand, QT performs quantizing the weights during the training process and training a DNN model with quantized weights during forward and backward propagations (Li et al., 2017a,b; Choi et al., 2019). We investigate the effect of QT that involves quantization during the training process on the inference accuracy. **Figures 8A,B** show simulated classification accuracies of QNN using PTQ for CIFAR10

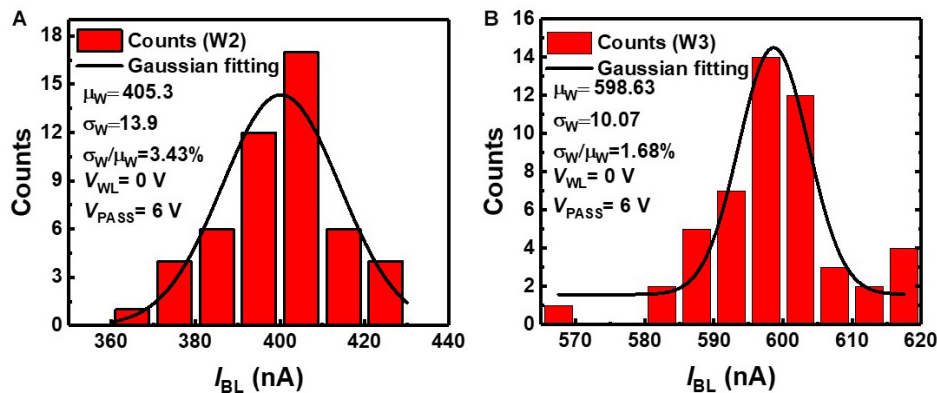


FIGURE 5 | I_{BL} distribution of NAND cells in NAND array at (A) W2 and (B) W3 levels.

and MNIST, respectively. Classification accuracies decrease by 0.33 and 1.26% with PTQ for MNIST and CIFAR10 images, respectively, compared with those obtained from neural networks having floating-point weight, as the bit-width of weight decreases to 4. Therefore, the PTQ scheme significantly decreases inference accuracy with 4-bit weight.

To decrease the degradation of classification accuracy, we adopt QT, which is an algorithm that involves fine-tuning optimized for QNN. **Figure 9** shows the simulated classification accuracy of neural networks using QT. QT increases classification accuracies by 0.34 and 0.96% for MNIST and CIFAR10, respectively, compared with those for PTQ. The classification accuracies using QT for MNIST and CIFAR10 are 98.2 and 89.7%, respectively, which are comparable with those obtained in neural networks having floating-point weight (FNN), as shown in the inset. Therefore, by adopting QT, the neuromorphic system utilizing NAND flash memory weighting 4-bit can achieve high inference accuracy. The power efficiency of the synaptic device is estimated from the distribution of synaptic weights in QNN. The average power consumed in a synaptic device per neural computation is estimated to be 0.15 μ W for multilayer neural

networks consisting of five layers (784–1024–1024–1024–10). The power consumption of the synaptic device can be reduced by adopting a thin (~ 3 nm) body (Lue et al., 2019) or pruning the neural networks (Lee et al., 2020). Note that, in this work, we use a 4-bit weight because a 4-bit weight can achieve higher accuracy than binary weight and achieve comparable accuracy compared with a 6-bit weight (Hubara et al., 2017). If a synaptic device has a 5-bit conductance level to implement a 6-bit weight, more time and energy are required in the RVW process for weight transfer.

To investigate the effect of weight and input precision on the classification accuracy of the neural networks, QNN, having 4-bit weight and analog input, is compared with binary neural networks (BNN) having 1-bit weight and 1-bit input. **Figure 10** shows the inference accuracy of QNN and BNN for CIFAR10 with convolution neural networks having three fully connected layers and six convolution layers. Note that, as bit-width of weight and input in QNN decreases, the classification accuracy decreases (Hubara et al., 2017). It is because the quantization of weights and inputs results in a weighted sum error. In addition, the reduction of bit-width of quantization increases the weighted sum error, which decreases classification accuracy. The final classification accuracies are 89.38 and 87.1% for QNN and BNN, respectively. Therefore, the proposed operation scheme can implement QNN with higher inference accuracy compared with BNN (Lee et al., 2019a).

Effect of Device Non-ideality

Figure 11 shows the effect of device variation (σ_w/μ_w) on simulated classification accuracy of QNN for CIFAR10 and MNIST images. The simulation is executed 20 times at each σ_w/μ_w , assuming a Gaussian distribution (Lee et al., 2019b). The classification accuracy decreases as the device variation increases. In this work, the largest device variation among eight levels is 3.43% (W2), so it is used to estimate the classification accuracy. As the device variation (σ_w/μ_w) of our work is sufficiently low, the inference accuracies decrease by less than 0.16 and 0.24% for the MNIST and CIFAR 10 images, respectively, compared with accuracy with no variation. To reduce the variation in the conductance of synaptic devices, it is necessary to reduce

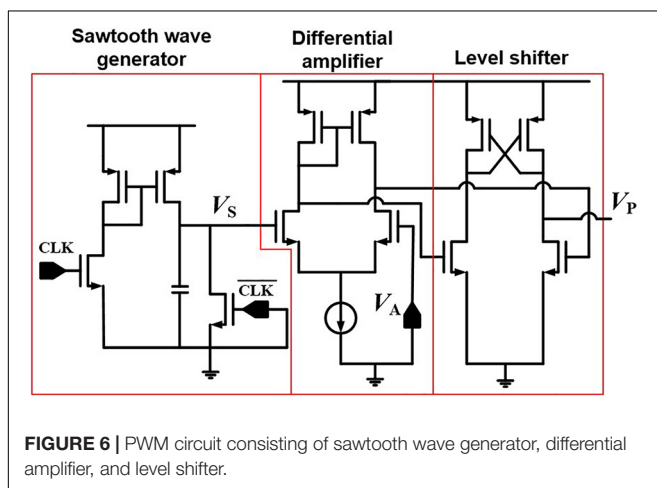


FIGURE 6 | PWM circuit consisting of sawtooth wave generator, differential amplifier, and level shifter.

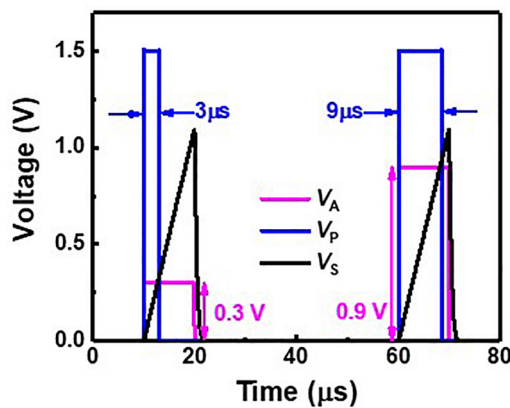


FIGURE 7 | Simulated result of V_A , V_S , and V_P in PWM circuit.

the target current range set in the control circuits of the RVW method. However, this increases the number of pulses applied to devices, which increases energy and time consumption in the RVW process. Therefore, it is necessary to set the optimized target current range in RVW, taking into account the degree of conductance variation and the energy and time consumed in the

RVW process. The variation obtained in this work is less than 3.43%, which is sufficiently low to achieve comparable accuracy compared with that with no variation.

Figure 12 shows the effect of the stuck-at-off device ratio on simulated classification accuracy of QNN for CIFAR10 and MNIST images. The simulation is executed 20 times at each ratio, and the classification accuracy decreases as the ratio of stuck-at-off device increases. The classification accuracies decrease by 13.5 and 0.5% for CIFAR10 and MNIST, respectively, as the stuck-at-off device ratio increases to 10%. To reduce degradation of classification accuracy due to the stuck-at-off device below 1% for CIFAR10, the ratio of the stuck-at-off device needs to be below 2%. NAND flash memory is currently a mass-produced technology, and the ratio of stuck-at-off cells is estimated to be less than 1%.

DISCUSSION

Comparison of Input Pulse Schemes

To implement VMM in a neuromorphic system, the intensity of the input signal in the DNN algorithm can be represented by the amplitude or width of the input pulse. However, the amplitude modulation scheme causes an error in VMM because

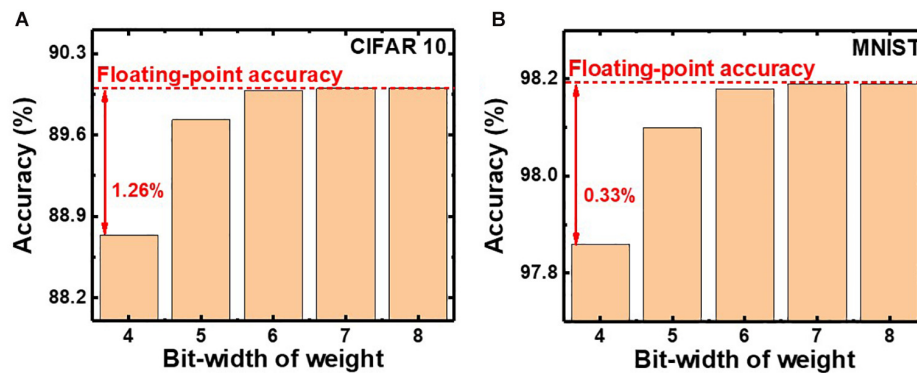


FIGURE 8 | Simulated classification accuracy with respect to the bit-width of weight using PTQ for (A) CIFAR10 and (B) MNIST images.

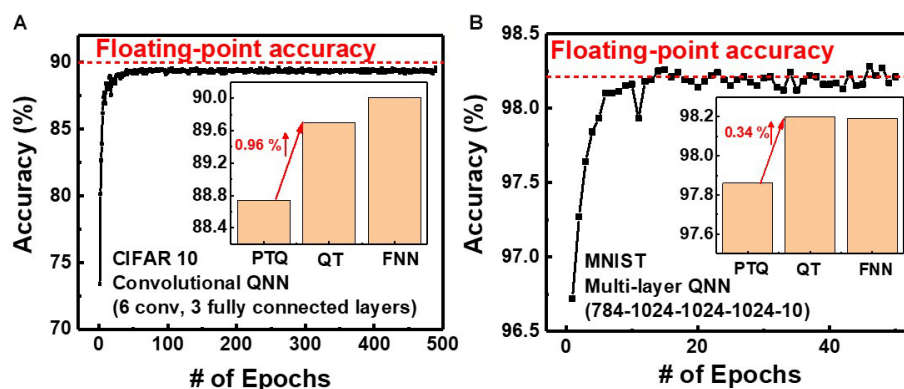


FIGURE 9 | Simulated classification accuracy with QT for (A) CIFAR10 and (B) MNIST images.

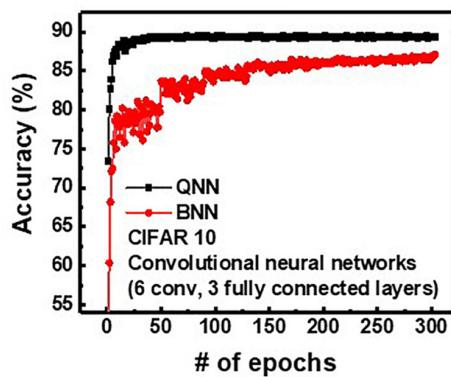


FIGURE 10 | Simulated classification accuracy of QNN and BNN for CIFAR10 images.

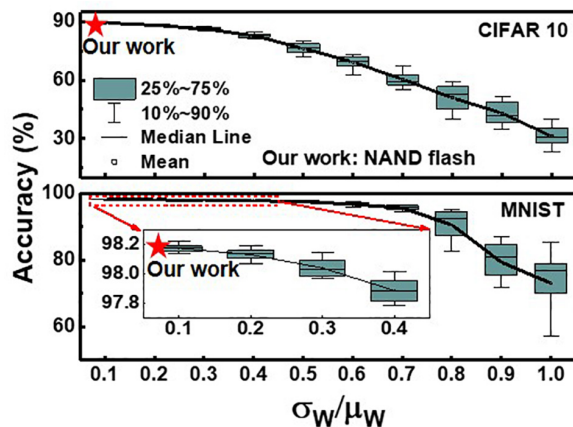


FIGURE 11 | Effect of device variation (σ_w/μ_w) on simulated classification accuracy of QNN for CIFAR 10 and MNIST images. Red star represents the accuracy when the largest variation obtained in this work is applied.

the I - V characteristics of synaptic devices are non-linear (Kim T. et al., 2017). To resolve this problem, a previous study reported an input pulse mapping scheme using an inverse function

generator that handles the non-linearity of I - V characteristics (Kim T. et al., 2017). This solves the non-linearity problem, but the VMM can still be inaccurate due to unwanted voltage drop across the parasitic resistance of the pass cells or metal wire. As described earlier, the amplitude modulation scheme has limitations in realizing accurate VMM operation but can reduce latency compared with the width modulation scheme.

On the other hand, the width modulation scheme can eliminate the effect of parasitic resistance by operating synaptic devices in the saturation region of I - V characteristics. This scheme may have a longer latency than the amplitude modulation scheme but enables accurate VMM. The width modulation scheme requires a PWM circuit to convert the intensity of the input to the width of the input pulse, which increases the burden on the peripheral circuit. Because the amplitude modulation scheme requires an inverse function generator that requires an operational amplifier, it also increases the burden on the peripheral circuit (Kim T. et al., 2017).

Comparison With Prior Works

In prior studies, our group has reported neuromorphic architectures that use NAND flash memory cells as binary synapses performing XNOR operation in BNNs (Lee et al., 2019a) and synaptic devices in on-chip training (Lee et al., 2018). In those studies, output current for each neuron is sequentially generated each time V_{read} is imposed on a selected WL. However, in this work, all outputs of neurons in a neuron layer are generated in a single input pulse. In addition, in the previous study of Lee et al. (2018), the conductance of synaptic devices is changed by applying an identical pulse to the synaptic device in on-chip learning. In this study, the conductance of synapse is tuned by the RVW method in off-chip learning. In Lee et al. (2019a), binary synaptic architecture capable of XNOR operation digitally was reported. However, this work proposes the VMM of multi-bit input and multi-bit weight in an analog fashion, significantly decreasing the burden of neuron circuits compared with the scheme of digital fashion.

A design scheme of synaptic architecture using NAND flash memory for performing MAC with multi-bit weight and multi-bit input has been proposed in Lue et al. (2019). In this

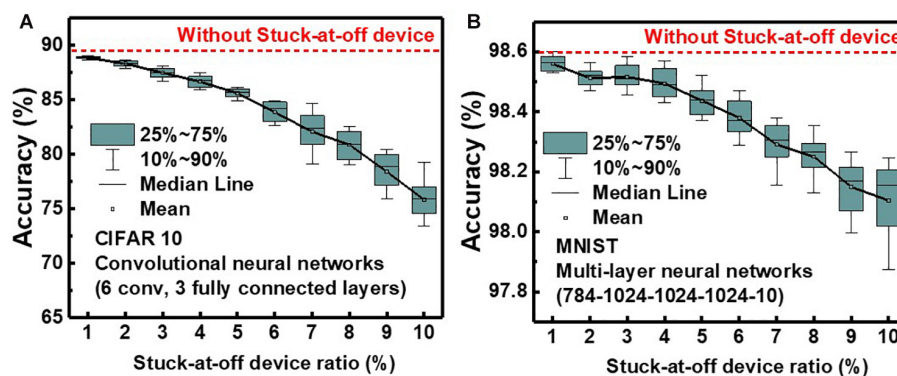


FIGURE 12 | Effect of stuck-at-off device ratio on simulated classification accuracy of QNN for (A) CIFAR 10 and (B) MNIST images.

scheme, lots of binary cells and BLs are utilized to represent a multilevel weight and a multilevel input, respectively, resulting in a substantial disadvantage in terms of synapse density (Lue et al., 2019). Furthermore, “shifter and adder” design is utilized to generate multilevel MAC, resulting in lots of burden in peripheral circuits (Lue et al., 2019). On the other hand, the proposed scheme in this work uses two NAND cells as one synaptic device and utilizes the PWM circuit to represent multi-bit input, which significantly increases the density of synaptic devices. Furthermore, the VMM can be performed in a pulse step using the proposed scheme in this work, greatly reducing the CMOS overhead in peripheral circuits compared with the “shifter and adder” design.

CONCLUSION

We have proposed a novel operating method and architecture for neuromorphic computing using PWM in the NAND flash memory architecture and evaluated its performance. The proposed operation scheme is well fitted to conventional NAND flash memory to implement QNNs with width-modulated input pulse and 4-bit weight. In addition, VMM of analog input and 4-bit weight can be implemented with a single pulse without additional logic operation. By utilizing a RVW scheme, eight conductance levels from 0 to 1.4 μA were demonstrated with a device variation of less than 3.43%. QT increases accuracies by

0.34 and 0.96% for MNIST and CIFAR10 images, respectively, compared with PTQ. Sufficiently low device variation (3.43%) of NAND cells results in high inference accuracy. Finally, the proposed operation scheme in this work can implement high-density, highly robust, and highly efficient neuromorphic systems using NAND flash memory architecture.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusion of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

S-TL and J-HL conceived and designed the experiments and wrote the manuscript. S-TL performed the simulation for MNIST and CIFAR10 classification, theoretical analyses, and measured device characteristics. All authors discussed the results and commented on the manuscript.

FUNDING

This work was supported by the National Research Foundation of Korea (NRF-2016M3A7B4909604) and the Brain Korea 21 Plus Project in 2020.

REFERENCES

- Andri, R., Cavigelli, L., Rossi, D., and Benini, L. (2018). “YodaNN: an architecture for ultralow power binary-weight CNN acceleration,” in *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Piscataway, NJ. doi: 10.1109/TCAD.2017.2682138
- Choi, J., Venkataramani, S., Srinivasan, V., Gopalakrishnan, K., Wang, Z., and Chuang, P. (2019). “Accurate and efficient 2-bit quantized neural networks,” in *Proceedings of the 2nd SysML Conference*, Boston, FL.
- Gao, L., Chen, P.-Y., and Shimeng, Y. (2016). Demonstration of convolution kernel operation on resistive cross-point array. *IEEE Electron Dev. Lett.* 37, 870–873. doi: 10.1109/led.2016.2573140
- Guan, Y., and Ohsawa, T. (2019). “Co-design of DNN model optimization for binary ReRAM array in-memory processing,” in *Proceedings of the 2019 IEEE 11th International Memory Workshop (IMW)*, Monterey, CA. doi: 10.1109/IMW.2019.8739722
- 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.
- Huh, H., Cho, W., Lee, J., Noh, Y., Park, Y., Ok, S., et al. (2020). “A 1Tb 4b/Cell 96-stacked-WL 3D NAND flash memory with 30MB/s program throughput using peripheral circuit under memory cell array technique,” in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA. doi: 10.1109/ISSCC19947.2020.9063117
- Jackson, B. L., Rajendran, B., Corrado, G. S., Breitwisch, M., Burr, G. W., Cheek, R., et al. (2013). Nanoscale electronic synapses using phase change devices. *ACM J. Emerg. Technol. Comput. Syst.* 9, 1–20. doi: 10.1201/9780367808624-1
- Kang, D., Kim, M., Jeon, S. C., Jung, W., Park, J., Choo, G., et al. (2019). “13.4 A 512Gb 3-bit/Cell 3D 6th-Generation V-NAND flash memory with 82MB/s write throughput and 1.2 Gb/s interface,” in *Proceedings of the 2019 IEEE International Solid-State Circuits Conference (ISSCC)*, New York, NY. doi: 10.1109/ISSCC.2019.8662493
- Khan, S. H., Hayat, M., and Porikli, F. (2019). Regularization of deep neural networks with spectral dropout. *Neural Netw.* 110, 82–90. doi: 10.1016/j.neunet.2018.09.009
- Kim, H., Hwang, S., Park, L., and Park, B.-G. (2017). Silicon synaptic transistor for hardware-based spiking neural network and neuromorphic system. *Nanotechnology* 28:40. doi: 10.1088/1361-6528/aa86f8
- Kim, T., Kim, H., Kim, J., and Kim, J. J. (2017). Input voltage mapping optimized for resistive memory-based deep neural network hardware. *IEEE Electron Dev. Lett.* 38, 1228–1231. doi: 10.1109/led.2017.2730959
- Lee, S. T., Kim, H., Bae, J., Yoo, H., Choi, N., Kwon, D., et al. (2019a). “High-density and highly-reliable binary neural networks using NAND flash memory cells as synaptic devices,” in *Proceedings of the 2019 IEEE Int. Electron Devices Meeting (IEDM)*, San Francisco, CA. doi: 10.1109/IEDM19573.2019.8993478
- Lee, S. T., Lim, S., Choi, N., Bae, J., Kwon, D., Park, B., et al. (2019b). Operation scheme of multi-layer neural networks using NAND flash memory as high-density synaptic devices. *IEEE J. Electron Dev. Soc.* 7, 1085–1093. doi: 10.1109/jeds.2019.2947316
- Lee, S. T., Lim, S., Bae, J. H., Kwon, D., Kim, H. S., Park, B. G., et al. (2020). Pruning for hardware-based deep spiking neural networks using gated schottky diode as synaptic devices. *J. Nanosci. Nanotechnol.* 20, 6603–6608. doi: 10.1166/jnn.2020.18772
- Lee, S. T., Lim, S., Choi, N., Bae, J. H., Kim, C. H., Lee, S., et al. (2018). “Neuromorphic technology based on charge storage memory devices,” in *Proceedings of the IEEE Symposium on VLSI Technology*, Honolulu, HI. doi: 10.1109/VLSIT.2018.8510667
- Li, H., De, S., Xu, Z., Studer, C., Same, H., and Goldstein, T. (2017a). “Towards a deeper understanding of training quantized neural networks,” in *Proceedings of the ICML 2017 Workshop on Principled Approaches to Deep Learning (PADL)*, Sydney.
- Li, H., De, S., Xu, Z., Studer, C., Samet, H., and Goldstein, T. (2017b). “Training quantized nets: a deeper understanding,” in *Proceedings of Advances in Neural Information Processing Systems*, 5811–5821.

- Lue, H. T., Hsu, P. K., Wei, M. L., Yeh, T. H., Du, P. Y., Chen, W. C., et al. (2019). "Optimal design methods to transform 3D NAND flash into a high-density, high-bandwidth and low-power nonvolatile computing in memory (nvCIM) accelerator for deep-learning neural networks (DNN)," in *Proceedings of the International Electron Device Meeting (IEDM)*, San Francisco, CA. doi: 10.1109/IEDM19573.2019.8993652
- Nishani, E., and Cico, B. (2017). "Computer vision approaches based on deep learning and neural networks: Deep neural networks for video analysis of human pose estimation," in *Proceedings of the 2017 6th Mediterranean Conference on Embedded Computing (MECO)*, Bar. doi: 10.1109/MECO.2017.7977207
- Park, S., Sheri, A., Kim, J., Noh, J., Jang, J., Jeon, M., et al. (2013). "Neuromorphic speech systems using advanced ReRAM-based synapse," in *Proceedings of the 2013 IEEE International Electron Devices Meeting*, Washington, DC. doi: 10.1109/IEDM.2013.6724692
- Sainath, T. N., Weiss, R. J., Wilson, K. W., Li, B., Variiani, E., Bacchiani, M., et al. (2017). "Multichannel signal processing with deep neural networks for automatic speech recognition," in *Proceedings of the IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Piscataway, NJ. doi: 10.1109/TASLP.2017.2672401
- Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. (2017). Group sparse regularization for deep neural networks. *Neurocomputing* 241, 81–89. doi: 10.1016/j.neucom.2017.02.029
- Sun, X., Peng, X., Chen, P.-Y., Liu, R., Seo, J.-S., and Yu, S. (2018). "Fully parallel RRAM synaptic array for implementing binary neural network with (+1, −1) weights and (+1, 0) neurons," in *Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju. doi: 10.1109/ASPDAC.2018.8297384
- Suri, M., Bichler, O., Querlioz, D., Cueto, O., Perniola, L., Sousa, V., et al. (2011). "Phase change memory as synapse for ultra-dense neuromorphic systems: application to complex visual pattern extraction," in *Proceedings of the 2011 International Electron Devices Meeting*, Washington, DC. doi: 10.1109/IEDM.2011.6131488
- Tang, T., Xia, L., Li, B., Wang, Y., and Yang, H. (2017). "Binary convolutional neural network on RRAM," in *Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba. doi: 10.1109/ASPDAC.2017.7858419
- Truong, L., Barik, R., Totoni, E., Liu, H., Markley, C., Fox, A., et al. (2016). ". Latte: a language, compiler, and runtime for elegant and efficient deep neural networks," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY. doi: 10.1145/2908080.2908105
- Wang, C., Feng, D., Tong, W., Liu, J., Wu, B., Zhao, W., et al. (2020). "Improving write performance on cross-point RRAM arrays by leveraging multidimensional non-uniformity of cell effective voltage," in *Proceedings of the IEEE Transactions on Computers*, Piscataway, NJ. doi: 10.1109/TC.2020.2990884
- Woo, J., and Yu, S. (2019). "Impact of selector devices in analog RRAM-based crossbar arrays for inference and training of neuromorphic system," in *Proceedings of the IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, London. doi: 10.1109/TVLSI.2019.2917764
- Yamashita, R., Magia, S., Higuchi, T., Yoneya, K., Yamamura, T., Mizukoshi, H., et al. (2017). "A 512gb 3b/cell flash memory on 64-word-line-layer bics technology," in *Proceeding of the 2017 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA. doi: 10.1109/ISSCC.2017.7870328
- Yu, S., Sun, X., Peng, X., and Huang, S. (2020). "Compute-in-memory with emerging nonvolatile-memories: challenges and prospects," in *Proceedings of the 2020 IEEE Custom Integrated Circuits Conference (CICC)*, Boston, MA. doi: 10.1109/CICC48029.2020.9075887
- Zhou, Z., Huang, P., Xiang, Y. C., Shen, W. S., Zhao, Y. D., Feng, Y. L., et al. (2018). "A new hardware implementation approach of BNNs based on nonlinear 2T2R synaptic cell," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA. doi: 10.1109/IEDM.2018.8614642

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 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.



Unsupervised Adaptive Weight Pruning for Energy-Efficient Neuromorphic Systems

Wenzhe Guo^{1,2}, Mohammed E. Fouda³, Hasan Erdem Yantir^{1,2}, Ahmed M. Eltawil^{2,3} and Khaled Nabil Salama^{1*}

¹ Sensors Lab, Advanced Membranes & Porous Materials Center, Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia, ² Communication and Computing Systems Lab, Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia, ³ Department of Electrical Engineering and Computer Science, University of California, Irvine, Irvine, CA, United States

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Anup Das,
Drexel University, United States
Abhronil Sengupta,
Pennsylvania State University (PSU),
United States

*Correspondence:

Khaled Nabil Salama
khaled.salama@kaust.edu.sa

Specialty section:

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

Received: 25 August 2020

Accepted: 26 October 2020

Published: 12 November 2020

Citation:

Guo W, Fouda ME, Yantir HE,
Eltawil AM and Salama KN (2020)
Unsupervised Adaptive Weight
Pruning for Energy-Efficient
Neuromorphic Systems.
Front. Neurosci. 14:598876.
doi: 10.3389/fnins.2020.598876

To tackle real-world challenges, deep and complex neural networks are generally used with a massive number of parameters, which require large memory size, extensive computational operations, and high energy consumption in neuromorphic hardware systems. In this work, we propose an unsupervised online adaptive weight pruning method that dynamically removes non-critical weights from a spiking neural network (SNN) to reduce network complexity and improve energy efficiency. The adaptive pruning method explores neural dynamics and firing activity of SNNs and adapts the pruning threshold over time and neurons during training. The proposed adaptation scheme allows the network to effectively identify critical weights associated with each neuron by changing the pruning threshold dynamically over time and neurons. It balances the connection strength of neurons with the previous layer with adaptive thresholds and prevents weak neurons from failure after pruning. We also evaluated improvement in the energy efficiency of SNNs with our method by computing synaptic operations (SOPs). Simulation results and detailed analyses have revealed that applying adaptation in the pruning threshold can significantly improve network performance and reduce the number of SOPs. The pruned SNN with 800 excitatory neurons can achieve a 30% reduction in SOPs during training and a 55% reduction during inference, with only 0.44% accuracy loss on MNIST dataset. Compared with a previously reported online soft pruning method, the proposed adaptive pruning method shows 3.33% higher classification accuracy and 67% more reduction in SOPs. The effectiveness of our method was confirmed on different datasets and for different network sizes. Our evaluation showed that the implementation overhead of the adaptive method regarding speed, area, and energy is negligible in the network. Therefore, this work offers a promising solution for effective network compression and building highly energy-efficient neuromorphic systems in real-time applications.

Keywords: neuromorphic computing, spiking neural networks, pruning, unsupervised learning, STDP, pattern recognition

INTRODUCTION

In recent years, as the prediction of Moore's law slows down prominently, neuromorphic computing has been widely regarded as a promising approach for large-scale computing. Neuromorphic systems are constructed following biological principles existing in our central nervous systems, which features in massive parallelism, collocated memory, and processors, and asynchronous event-driven computation (Mead, 1990; Furber et al., 2014; Davies et al., 2018).

Generally considered as the third generation of neural network models, spiking neural networks (SNNs) have started a paradigm shift in the brain-inspired research exploration. Different from artificial neural networks (ANNs), SNNs are well-known for its capability of accurately capturing neural dynamics and biological behaviors of the central nervous system and processing spatio-temporal information. With energy-efficient computation and parallel information processing features, SNNs are widely adopted for building neuromorphic hardware systems (Thakur et al., 2018). In such systems, information is transmitted through synapses from a presynaptic neuron to a postsynaptic neuron on the occurrence of an event (or a spike). Neural networks require deep and complex structures to tackle real-world tasks, like pattern recognition, object detection, and motor controls (Shrestha and Mahmood, 2019). The complexity leads to large synaptic memories and high energy consumption, which poses a big challenge in hardware implementation. Therefore, it is necessary to search for practical solutions to reduce network complexity and improve the energy efficiency of SNNs.

During early brain development, creations of synaptic connections between neurons exponentially increase with the numerous stimuli coming from environments every day (Zillmer and Spiers, 2001). The rapid synapse creation is vital for learning and memory formation. Between early childhood and adulthood, weight pruning occurs as a natural process during which our brain eliminates unnecessary synaptic connections. It is regarded as a purposeful process of maintaining a more efficient brain function. This biological process has been extensively studied in current ANNs for its attractive memory and energy reduction benefits. Han et al. (2015) introduced a training-pruning-retraining approach that can reduce the number of synaptic connections by 12x and computational operations by 5x for the VGG-16 network. Weight pruning was also proved to be an effective means of alleviating the overfitting problem in ANNs (Paupamah et al., 2020). Moreover, to avoid irregular structure of pruned weight matrices and aid in the leverage of sparse matrix-vector multiplication, a variety of structured weight pruning techniques were proposed where the entire rows and columns in the weight matrices are removed by imposing certain constraints during the pruning process (Anwar et al., 2017; Sredojevic et al., 2017).

While weight pruning has been widely applied in different ANNs, the benefits that weight pruning could provide for SNNs have yet to be explored. Limited works have reported applying weight pruning in SNNs so far (Iglesias et al., 2005; Rathi et al., 2019; Shi et al., 2019). Rathi et al. proposed a spike-timing-dependent plasticity (STDP) based online synaptic

pruning method, which sets non-critical weights to zero during the training phase and removes the weights below a certain threshold at the end of training (Rathi et al., 2019). This method only sets the weights to zero without removing them. It allows them to be updated during training, which is not an effective approach to improve the energy efficiency for online learning systems. Shi et al. presented an online soft-pruning method by setting the weights below a constant threshold to a constant value instead of removing them during training. While this method could reduce the number of STDP updates during training, it does not induce any sparsity in the network, leading to little benefit for hardware implementation. Moreover, these pruning methods use a constant weight threshold throughout the whole pruning process. With a constant threshold, the network can not effectively select the non-critical weights to be pruned. In the early phase of training, weights are not completely learned, and a large threshold can mistakenly remove important weights. If a small threshold is used, some non-critical weights can not be pruned at the end of training since these weights could grow. On the other hand, the connection strength of neurons in one layer with the previous layer varies. A large threshold could remove most of the critical weights from the neurons with weak connection and hence severely affect the neurons' function, which could lead to substantial performance degradation of the network. Therefore, it is crucial to adapt the weight threshold over time and all the neurons during training to improve network performance. In this work, we propose an online adaptive weight pruning method that adapts the pruning threshold over time and neurons during training and completely remove the weights below the threshold from the network. It is demonstrated to be an effective approach for reducing network complexity and improving energy efficiency during both training and inference operations.

The main contributions of this work are summarized as follows.

- A simple online adaptation scheme for the pruning threshold is presented, which can change the threshold dynamically over time during training. It also considers the spatial difference of the connection strength of neurons in one layer with the previous layer and adapts the threshold over the neurons based on their firing activity.
- The proposed method is demonstrated to be more effective in retaining classification accuracy after pruning than the constant threshold weight pruning and neuron pruning methods. It resulted in a 67% reduction in synaptic operations (SOPs) while outperforming the previously reported soft weight pruning method by 3.33%. The advantage of the proposed method was confirmed in the SNN on different datasets with different network sizes.
- In terms of training, the proposed online adaptive pruning method outperformed post-training pruning methods by providing more than 30% reduction in training SOPs when the pruning percentage is larger than 90% while providing more than 3% higher classification accuracy, which shows significant potential for developing high-performance and energy-efficient online neuromorphic learning system.

- The overhead of implementing the proposed method in a neuromorphic system is demonstrated to be insignificant in terms of processing speed, area, and energy.

This paper is organized as follows: section “Methods and Results” introduces different neural models used in this work and the SNN architecture. It then presents an overview of our methods, algorithmic implementation details, and pruning results for each method. In section “Comparisons and Discussions,” different pruning methods are discussed and compared. Section “Conclusion” concludes this work.

METHODS AND RESULTS

Network Models and Architecture

To model spiking neurons, the leaky integrated-and-fire (LIF) model was used in this work because of its computational efficiency and capability of capturing the essential features of information processing in the nervous system (Burkitt, 2006). The model consists of one first-order linear differential equation that defines the dynamics of membrane potential where synapses are modeled as conductance, as described by

$$\tau_m \frac{dv}{dt} = (v_r - v) - g_e (v - E_{exc}) - g_i (v - E_{inh}) \quad (1)$$

where τ_m is the time constant, v_r is the resting membrane potential, g_e is the excitatory conductance associated with an excitatory channel, E_{exc} is the reverse potential of the channel, g_i is the conductance associated with an inhibitory channel, E_{inh} is the reverse potential of the channel. The model resets the membrane potential to v_r and generates a spike if the membrane potential reaches a defined threshold v_{th} . Synaptic conductance follows a time-varying dynamics governed by Diehl and Cook (2015),

$$\tau_g \frac{dg}{dt} = -g + \sum_j w_{ij} \delta(t - t_j^f) \quad (2)$$

where g is the conductance, τ_g is the time constant, w_{ij} is the synaptic weight from the presynaptic neuron j to the postsynaptic neuron i , and t_j^f is the firing time of the presynaptic neuron j .

Spike-timing-dependent plasticity relates the synaptic plasticity to the relative timing difference between a presynaptic spike and a postsynaptic spike. A triplet-based STDP model was used in this work because of its biological plausibility and easy implementation (Pfister and Gerstner, 2006). It overcomes the limitation of the paired-based STDP models to accommodate the dependence on the repetition frequency of the pairs of spikes. It was shown that the triplet rule is more biological plausible where its response can fit the experimental data from the visual cortical slices and hippocampal cultures (Pfister and Gerstner, 2006). The model considers sets of three spikes (one presynaptic and two postsynaptic spikes), each of which leaves a time-varying trace whose dynamics are described below.

$$\frac{ds}{dt} = -\frac{s}{\tau_s}, \quad s \in \{x_j, y_i^1, y_i^2\} \quad (3)$$

where x_j is the trace variable associated with the firing event of the presynaptic neuron j , y_i^1 , and y_i^2 are the fast and slow trace variables associated with the firing event of the postsynaptic neuron i , respectively, and τ_s is the corresponding time constant. When the presynaptic neuron (or postsynaptic) fires, the related trace x_j (or y_i) is reset to 1. The weight updates are carried out as below.

$$\Delta w_{ij} = \begin{cases} -\mu_{pre} y_i^1, & \text{if the neuron } i \text{ fires,} \\ +\mu_{post} x_j y_i^2, & \text{if the neuron } j \text{ fires.} \end{cases} \quad (4)$$

where μ_{pre} and μ_{post} are the corresponding weight updating rates.

In this work, a two-layer SNN architecture was adopted, as shown in **Figure 1**, and tested on the Modified National Institute of Standards and Technology (MNIST) dataset and Fashion-MNIST dataset (Lecun et al., 1998; Xiao et al., 2017). This architecture consists of an input layer and a processing layer. The input layer has 784 units, each of which receives the corresponding pixel in a digit image from the MNIST dataset and produces a Poisson spike train with a frequency proportional to the pixel intensity. This encoding scheme is commonly referred to as rate coding (O'Connor et al., 2013). The input layer is fully connected to the processing layer. In the processing layer, excitatory neurons send spikes to inhibitory neurons in a one-to-one fashion, whereas each inhibitory neuron sends spikes to all the excitatory neurons except the one that it receives spikes from. This connection pattern implements a winner-take-all (WTA) mechanism, which imposes lateral inhibition on excitatory neurons and hence competitions for learning input features. To ensure fair competition, a threshold adaptation scheme is applied. Whenever a neuron fires, its threshold is increased by an adaptation constant and then slowly decays with time. The phenomenon of threshold adaptation has been commonly observed in the central nervous system (Fontaine et al., 2014). In this work, the networks with 100 and 800 excitatory neurons were used to verify the effectiveness of our proposed pruning method. A simple classification scheme is implemented based on the firing activity of excitatory neurons. After training, excitatory neurons are assigned labels to which they fire the most spikes. They are then divided into ten groups, each of which corresponds to a digit and contains all the neurons labeled by this digit. During inference, the classification result for an input image is the digit of the group with the highest average spike counts. The model parameters used in the simulation are listed in **Table 1**. The parameters were configured through a genetic algorithm to achieve the best accuracy. The classification accuracy on MNIST dataset achieved in these two SNNs without pruning is 85.78%/90.40%, respectively, while the accuracy on Fashion-MNIST dataset is 64.57%/69.21%, respectively. All the simulations in this work were run in a Python-based platform.

Overview of the Proposed Pruning Methods

Pruning is a natural process existing in human brains to maintain their efficient function. It is widely adopted in neural networks to reduce network complexity and improve energy efficiency. Various works have demonstrated the practical effectiveness of

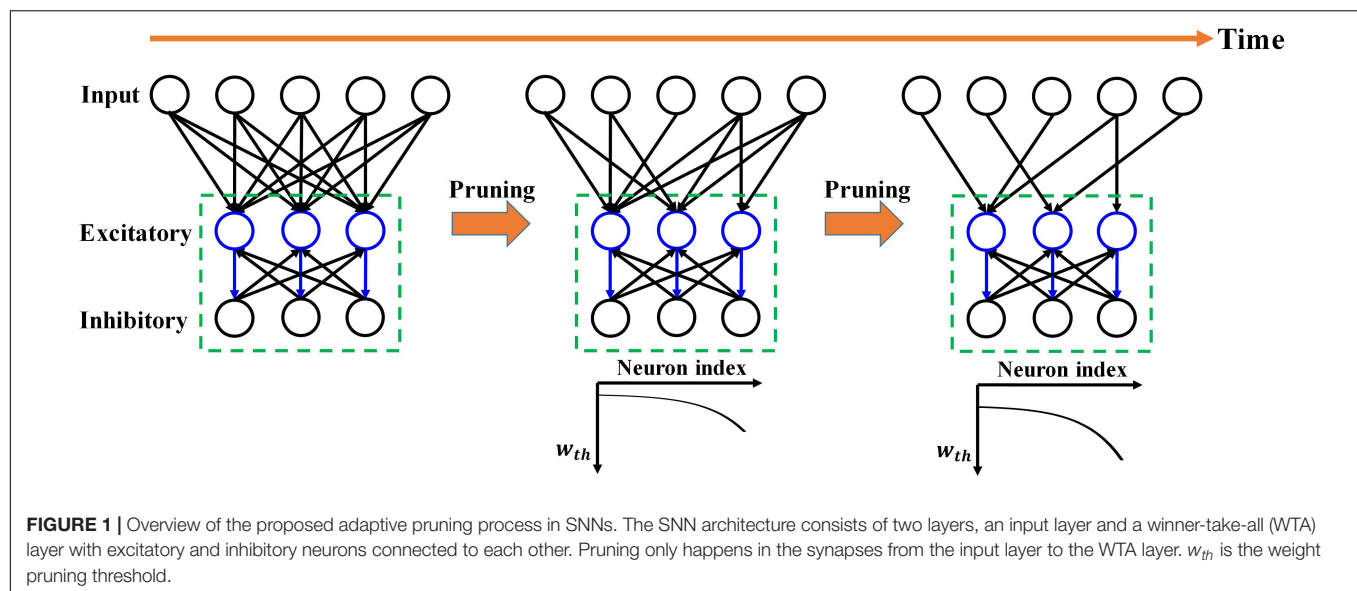


TABLE 1 | Model parameters used in the simulation.

Model parameters	Description	Value
$\tau_m, \tau_{ge}, \tau_{gi}$	Time constants in the LIF model	100 ms, 1 ms, 2 ms
$V_r, V_{th}, E_{exc}, E_{inh}$	Potential constants in the LIF model	-60 mV, -50 mV, 0, -100 mV
$\tau_x, \tau_{y1}, \tau_{y2}$	Time constants in the STDP model	8 ms, 16 ms, 32 ms
μ_{pre}, μ_{post}	Learning rates in the STDP model	0.0001, 0.01
θ	Threshold adaptation constant	0.01 mV

weight pruning in reducing the number of parameters and computational operations without losing accuracy (Han et al., 2015; Li et al., 2019; Tung and Mori, 2020). For example, Li et al. compressed different deep neural networks using weight pruning on mobile devices for real-time applications. They showed significant memory storage reduction and speedup. Pruning is commonly applied after training, which is suitable for improving the energy efficiency of inference systems with offline training. Pruning while training technique has been proved to be very useful in SNNs to improve online learning systems that can learn and infer the real-world information (Rathi et al., 2019; Shi et al., 2019).

However, in the previously reported online pruning methods, pruning was conducted with a constant threshold for all the synaptic weights during training. It is not an effective approach to select the non-critical weights since weights change over time during training. A large threshold can mistakenly remove many important weights at the beginning of training and severely affect the functions of the neurons with weak synaptic connections, leading to substantial performance degradation of the network, while a small threshold is not able to remove some non-critical weights at the end. In this work, we will present different pruning

while training methods by adapting the pruning threshold over time and neurons. The overview of the proposed pruning process is depicted in **Figure 1**, where the pruning process progresses during training. Pruning is carried out only in the synaptic weights are plastic and subject to training. Initially, the network has a fully-connected structure between the input layer and the WTA layer. When the pruning process starts, the pruning threshold (w_{th}) is adapted and remains different for all the excitatory neurons according to their firing activity. Fewer weights are removed for the neurons with lower thresholds. Moreover, over time, the threshold for each neuron is increased so that more weights are pruned at later pruning stages.

To perform pruning while training effectively, we need to determine when to start the pruning process. If the pruning process starts too early, important weights that have a profound impact on the output could be mistakenly pruned away, which will deteriorate network performance. On the other hand, if it starts too late, the network might not have enough training cycles to compensate for the accuracy loss and reduced improvement in training energy efficiency. To find this critical point, we have observed how the network dynamics evolve with time by monitoring firing activities and weight updates of excitatory neurons. **Figure 2A** shows that neurons start to fire regularly after training over 30,000 images, suggesting that the network has learned the major input features and starts to adjust for small details. In **Figure 2B**, the statistics (mean and variance) of weight updates over time have also revealed the same network behavior. As a result, the pruning process was decided to start after training over 30,000 images. Moreover, the pruning process was performed in multiple steps by dividing the whole dataset into multiple batches. The batch size was selected as 5,000 under the consideration of pruning frequency. The detailed implementation and algorithm of the proposed pruning methods are presented and discussed in the following sections.

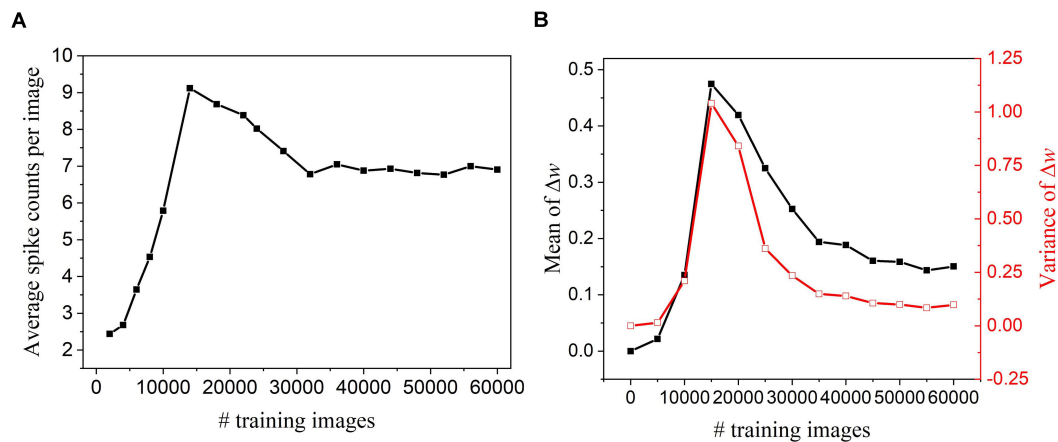


FIGURE 2 | Network dynamics were monitored every 5,000 training images in the SNN with 100 excitatory neurons during training and without pruning. **(A)** Firing activity. The average of spike counts of 100 excitatory neurons was calculated. **(B)** Statistics of weight updates Δw with the mean (black) and variance (red).

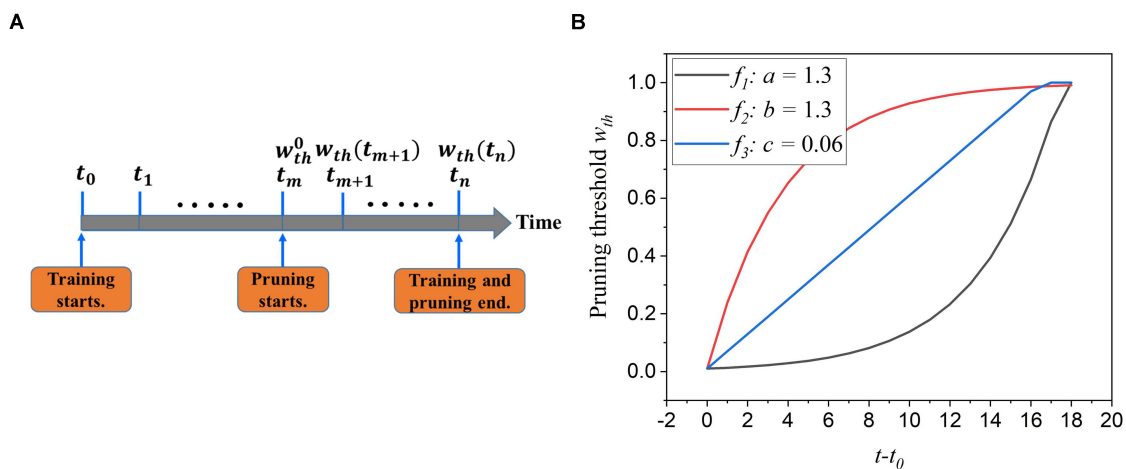


FIGURE 3 | (A) The illustration of the online adaptive pruning scheme over time. w_{th}^0 is the initial pruning threshold and $w_{th}(t)$ is the pruning threshold at time t . **(B)** The evolution of the pruning threshold over time with w_{th}^0 set as 0.036.

APT: Online Adaptive Pruning Over Time

During training, synaptic weights are randomly initialized and updated according to the input features over time. Some weights approach high value and contribute largely to network performance, whereas some are reduced to zero and less critical. The first adaptation scheme is to adapt the pruning threshold over time during training. The pruning process is illustrated in **Figure 3A**. The pruning starts after training over 30,000 images at a time t_m and an initial pruning threshold w_{th}^0 is given. It ends when the training process is finished. This scheme is to increase the pruning threshold with time. The motivation behind it is to allow more weights to be trained and avoid removing critical weights mistakenly at the early training phase. At the end of the training, weights are already trained enough, and a larger threshold will not significantly increase the chance of critical weights being pruned unintentionally. The threshold adaptation scheme can be formularized by $w_{th}(t) = f(t)$, where $w_{th}(t)$ is the

pruning threshold at time t , and $f(t)$ is the adaptation function. To select a suitable adaptation function, we propose two different exponential functions (f_1 and f_2) and a linear function (f_3), described below.

$$f_1(t) = w_{th}^0 a^{t-t_m}, \quad (5)$$

$$f_2(t) = w_{max} - (w_{max} - w_{th}^0) b^{-(t-t_m)}, \quad (6)$$

and

$$f_3(t) = w_{th}^0 + c(t - t_m) \quad (7)$$

where w_{th}^0 is the initial pruning threshold at the starting pruning time t_m , w_{max} is the maximum value of weights, a , b , and c are the corresponding adaptation factors in the functions. These three functions are all confined in the range $[w_{th}^0, w_{max}]$. It is worth noting that the pruning threshold has to be less than w_{max} to avoid pruning the entire network. **Figure 3B** shows how these

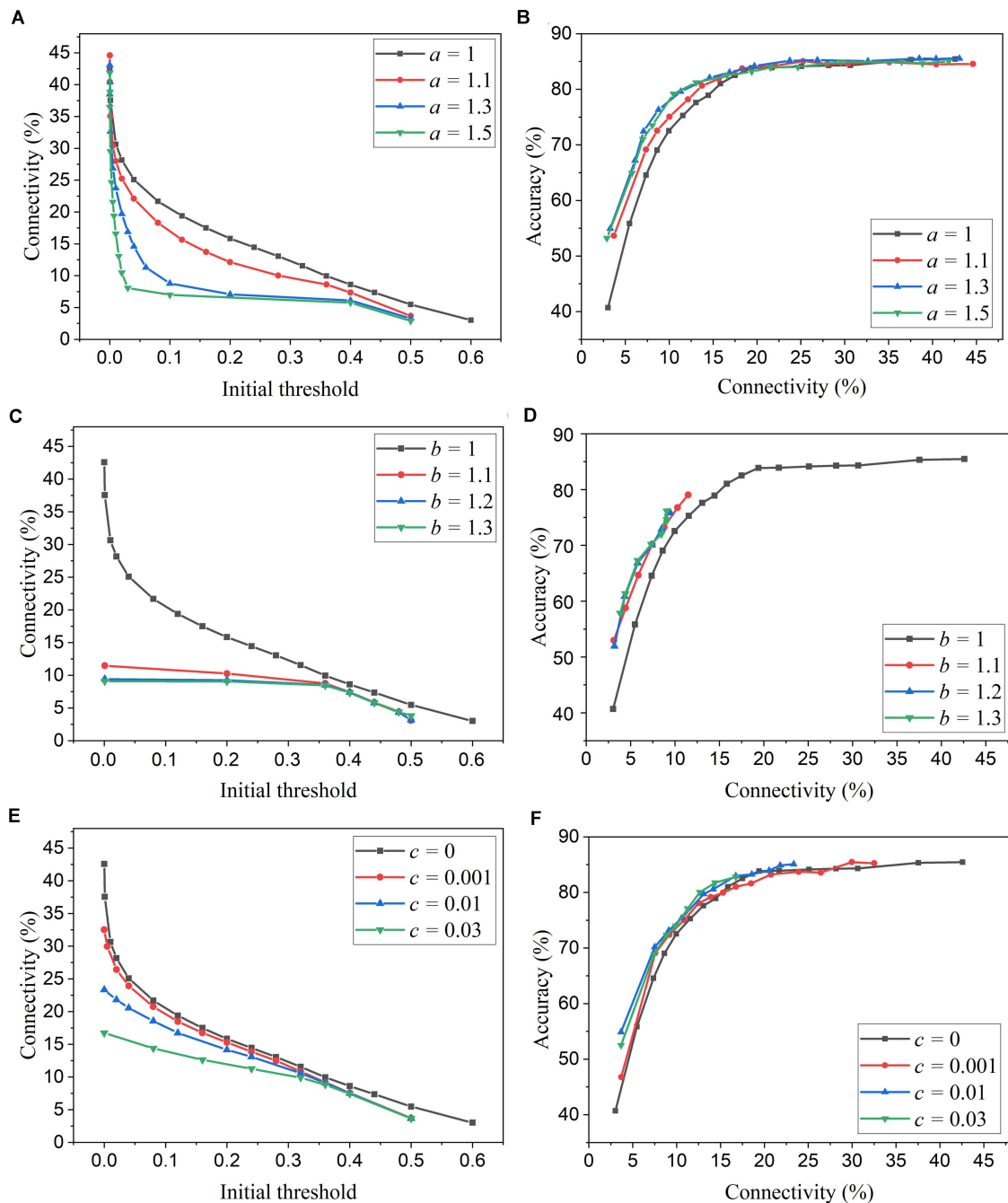
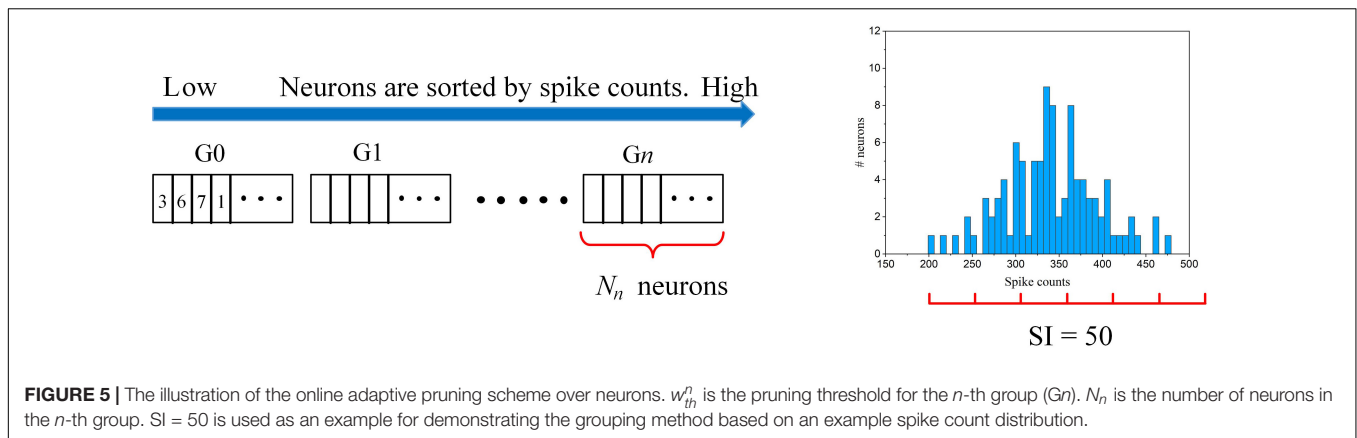


FIGURE 4 | Simulation results of the online adaptive pruning over time for different adaptation functions in the SNN trained on MNIST dataset with 100 excitatory neurons. (A,C,E) show the network connectivity changes with the initial threshold for the adaptation functions f_1 , f_2 , and f_3 , respectively. (B,D,F) present the accuracy changes with the network connectivity for the adaptation functions f_1 , f_2 , and f_3 , respectively. The connectivity is defined as the percentage of the unpruned weights in the total weights.

functions change the pruning threshold over time. Clearly, the f_1 function increases the threshold slowly at the beginning and rapidly at the end, while the f_2 function has the opposite effect. The linear function (f_3) keeps the same updating rate.

We simulated SNNs with the proposed online pruning method. Online pruning with a constant threshold was also

included as a reference to demonstrate the effectiveness of our proposed pruning methods and will be referred to as online constant pruning hereinafter. **Figure 4** shows the simulation results of SNNs with the online adaptive pruning over time for three different adaptation functions, namely f_1 , f_2 , and f_3 , as described above. For each adaptation function, the results of



network connectivity vs. initial threshold ($w_{th}(t_0)$) and accuracy vs. connectivity are presented, where the connectivity is defined as the percentage of non-zero weights in the total weights. The impact of different values of the corresponding adaptation factors (a , b , and c) is also studied. In **Figures 4A,C,E**, it can be seen that different initial thresholds result in different connectivity levels, and the higher the threshold, the smaller the connectivity. It should be noted that for $a = 1$, $b = 1$, and $c = 0$, they all are equivalent to the constant pruning case. By increasing the corresponding adaptation factors, a smaller initial threshold is needed to reach certain connectivity. In **Figure 4C**, with the factor $b > 1$, the connectivity becomes very small ($<15\%$) even when a very low threshold is used. This is because the adaptation function f_2 increases the threshold value very rapidly at the early phase of pruning process and hence results in a high threshold most of the time, as shown in **Figure 3B**. In **Figures 4B,D,F**, the accuracy decreases with the connectivity, as synaptic weights are pruned, and the network becomes sparse. We use the accuracy vs. connectivity as a performance metric to compare these adaptation functions and different pruning methods since pruning aims to reduce network complexity and maintain high classification accuracy. A network with high accuracy and small connectivity is desired. By applying adaptation over time (the adaptation factor >1), performance improvement can be observed for all the three adaptation functions. In **Figure 4B**, for the function f_1 , $a = 1.3$ is slightly better than $a = 1.5$ and hence selected as the optimized value for a . In **Figure 4D**, for the function f_2 , all the three cases (>1) have similar overall performance, but $b = 1.1$ is able to reach higher accuracy with larger connectivity and thus selected as the optimized value for b . In **Figure 4F**, for the function f_3 , $c = 0.01$ shows the best overall performance and thus is selected.

APN: Online Adaptive Pruning Over Neurons

The excitatory neurons in the network play different roles in contributing to network performance. The connection strength of these neurons to the input layer is different. A stronger connection makes the neuron more resilient to pruning, whereas a weaker connection makes the neuron more susceptible.

Applying a constant threshold for all the neurons can not effectively take the difference into consideration, and a large threshold can significantly deteriorate the function of the neurons with a weak connection. Thus, we propose an adaptation scheme over neurons by adapting the pruning threshold over all the excitatory neurons. The aim is to ensure that a smaller threshold is applied for weaker neurons and a larger threshold is for stronger neurons. So, we can balance the connection strength of all the neurons after pruning to achieve large network sparsity and maintain high classification accuracy.

The adaptation scheme is illustrated in **Figure 5**. The connection strength of each neuron to the input layer can be reflected by the firing activity. The more the neuron fires, the stronger connection it has to the input. So, the neurons are ranked according to their spike counts and divided into multiple groups. The spike count of each neuron was calculated as the average spike count during one batch training. Each group shares the same pruning threshold, and the threshold increases along from the first group (G_0) to the n -th group (G_n). The grouping scheme is explained as follows. Firstly, a spike count interval is defined as SI . Starting from the neuron with the minimum spike count, we group all the neurons with spike counts within $[S, S + SI]$, where S is the minimum spike count of the ungrouped neurons. In this way, the neurons in the same group have a spike count difference not larger than SI , so we can fairly sort the neurons with similar connection strength into one group. Across all the groups, the pruning threshold w_{th}^n is adapted according to an adaptation function $f(n)$, where n is the group index. In **Figure 5**, an example of the grouping process is shown where an example spike count distribution and $SI = 50$ are used. The neurons with spike counts that fall into an interval (red segment) are grouped together. In this example, six groups are sorted out. The number of neurons in each group is dependent on the spike count distribution and spike interval. The algorithmic implementation of this threshold adaptation scheme is described in **Algorithm 1**.

We simulated SNNs with the proposed online pruning method that uses the three different adaptation functions, namely f_1 , f_2 , and f_3 . Different values of the adaptation factor (a , b , and c) associated with each function were used in the simulation. The results are presented in **Figure 6**, including connectivity vs. initial

Algorithm 1 | Online adaptive pruning over time and neurons.

```

Input parameters: initial threshold  $w_{th}^0$ , spike count interval  $SI$ .
Pruning process starts after 30,000 training images.
Initialize the group index  $n = 1$ , the global threshold  $w_{th} = w_{th}^0$ .
for batch  $k = 1, 2, \dots$  do
  Sort all the excitatory neurons according to their spike counts from
  low to high.
  Set  $S = S_{min}$ , the minimum of the spike counts.
  for the sorted neuron  $i = 1, 2, \dots$  do
    if the spike count of the neuron  $i$ ,  $S_i < S + SI$ ,
      Set the pruning threshold of the neuron  $i$ ,  $w_{th}^i = w_{th}$ .
    else
      The group index  $n$  increases by 1.
      Adapt the pruning threshold,  $w_{th} = f_N(n)$ .
      Set  $S = S_i$ .
    end if
  end for
  Adapt the pruning threshold over time,  $w_{th} = f_T(k)$ .
end for

```

threshold and accuracy vs. connectivity. In the simulation, the spike count interval is fixed as 30 to study the impact of different adaptation functions. In **Figures 6A,C,E**, a higher threshold leads to smaller connectivity, and a larger adaptation factor requires a smaller threshold to reach certain connectivity. Similar observations to those described in the case of adaptation over time can also be seen. In **Figures 6B,D,F**, for the three different adaptation functions, the optimized values of the corresponding adaptation factors can be selected as $a = 1.15$, $b = 1.05$, and $c = 0.03$, respectively.

APT_N: Online Adaptive Pruning Over Time and Neurons

In the adaptation scheme over time, the threshold changes with time, but all the neurons share the same threshold. In contrast, the adaptation scheme over neurons considers the spatial difference of firing activities of all the neurons and adapts the threshold over them, but the thresholds for neuron groups are constant over time. A full adaptation scheme combines these two schemes by adapting the pruning threshold for each neuron over time during training. The algorithmic implementation is described in **Algorithm 1**. The pruning process starts after training over 30,000 images, and a global pruning threshold is initialized as w_{th}^0 . At each pruning step, all the neurons are sorted according to their spike counts in the order from low counts to high counts. Neurons are grouped according to the grouping method described in section “APN: Online Adaptive Pruning Over Neurons.” The global threshold is first assigned to the first neuron group (G0). It is then adapted according to the adaptation function $f_N(n)$, and assigned to the following groups. After the adaptation process over neurons is completed, the global threshold is reset and updated with the time adaptation function $f_T(k)$, where k is the pruning-step index. This combined method takes into consideration both the time evolution of synaptic weights and the spatial difference of firing activity of neurons during training.

COMPARISONS AND DISCUSSION

Comparison Among the Proposed Weight Pruning Methods

Firstly, we will compare the three different adaptation functions and select the best spike count interval. **Figure 7A** shows the comparison among the three adaptation functions with the optimized adaptation factors for the APT method. Clearly, the function f_1 gives the best performance improvement over the constant pruning method, i.e., the highest accuracy when connectivity is smaller than 15% and similar accuracy to other functions otherwise. This can be attributed to the fact that f_1 allows the pruning threshold to grow slowly at the early phase of the pruning process and hence more weights to be trained. It increases the threshold rapidly at the end, which guarantees largely reduced network connectivity. **Figure 7B** shows the comparison among the three adaptation functions with the optimized adaptation factors for the APN method. The same conclusion can be drawn that the function f_1 gives the best performance. Moreover, after selecting the adaptation function as f_1 , we studied the effect of the spike count interval on the performance. The results are shown in **Figure 8**. The spike count interval is used to identify how similar the firing activities of neurons in the same group are. In **Figure 8A**, with a smaller interval, a smaller initial threshold is needed to reach a certain threshold. A small interval results in a large number of groups and hence creates a large difference in pruning threshold among the groups. This can cause a very high threshold to be applied in the group with weak neurons and deteriorate their performance significantly. So, it is not an effective grouping. A large interval can gather the neurons with very different firing activity into one group where the same pruning threshold is shared. This way is also not effective because a large threshold can significantly deteriorate the performance of weak neurons and a small threshold is not able to remove enough non-critical weights from strong neurons. From the results in **Figure 8B**, $SI = 30$ shows the best performance.

To apply adaptation over both time and neurons, we combined the proposed adaptive pruning methods with the selected adaptation functions and adaptation factors. In this approach, the pruning threshold is increased over time and adapted across all the excitatory neurons. The comparisons among the proposed adaptive pruning methods for MNIST dataset and Fashion-MNIST dataset are shown in **Figures 9A,C** obtained from the SNN with 100 excitatory neurons, respectively. The same adaptation function and parameters were used to obtain the pruning results on Fashion-MNIST dataset. For all the pruning methods, up to 80% of weights trained on MNIST dataset can be pruned with less than 1% accuracy loss. It is because the trained weight maps on MNIST dataset are very sparse, as shown in **Figure 10A**. Whereas, only up to 50% of weights trained on Fashion-MNIST dataset can be pruned with less than 1% accuracy loss since the input patterns from Fashion-MNIST dataset are more complex, as shown in **Figure 10B**. Clearly, applying adaptation over both time and neurons can further improve the network performance, especially when the network becomes very sparse (connectivity < 10%). When the sparsity

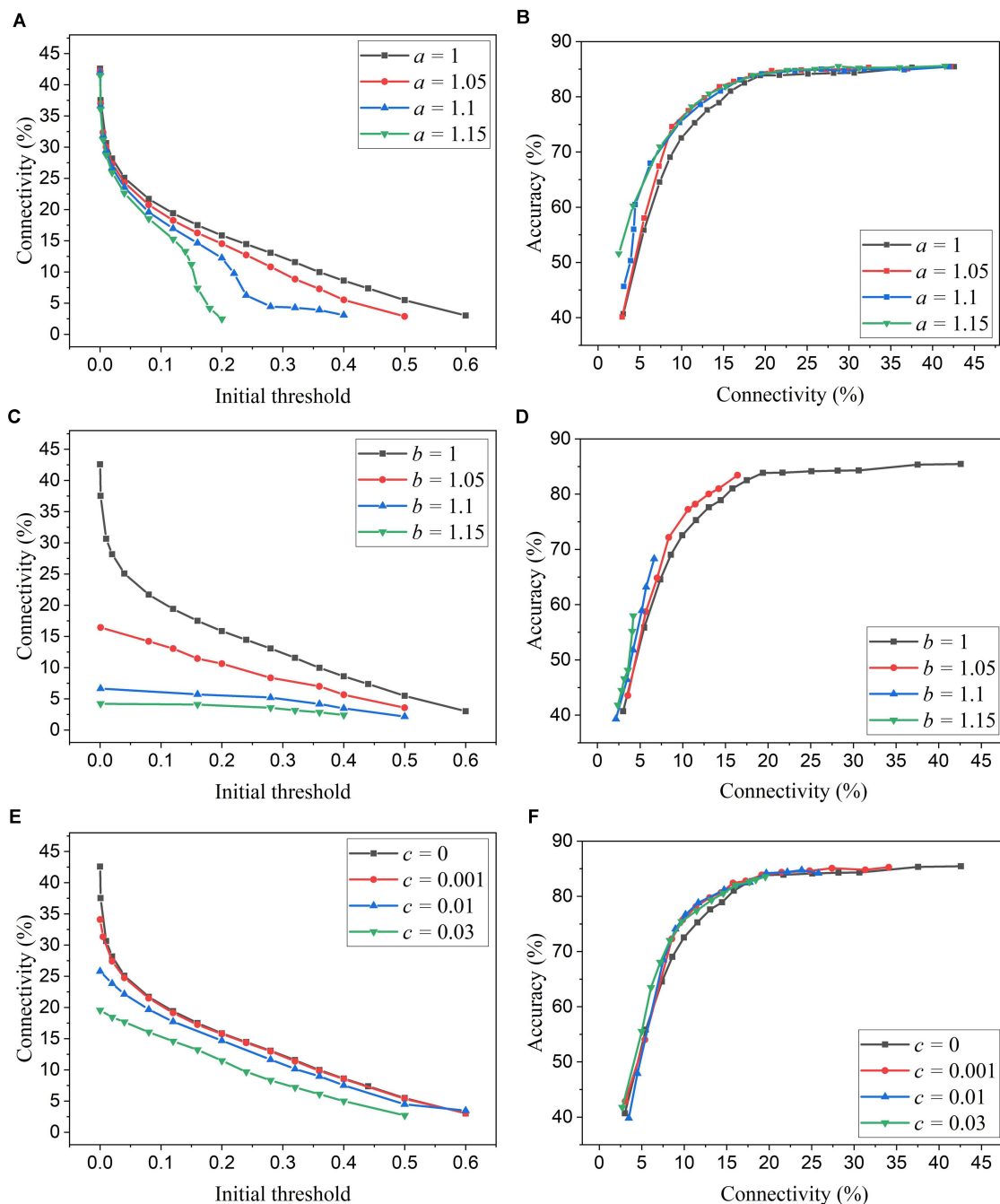


FIGURE 6 | Simulation results of the online adaptive pruning over neurons for different adaptation functions in the SNN trained on MNIST dataset with 100 excitatory neurons. **(A,C,E)** show the network connectivity changes with the initial threshold for the adaptation functions f_1 , f_2 , and f_3 , respectively. **(B,D,F)** present the accuracy changes with the connectivity for the adaptation functions f_1 , f_2 , and f_3 , respectively. The connectivity is defined as the percentage of the unpruned weights. The spike interval is set as 30.

of the network increases, the performance of each excitatory neuron is very sensitive to critical weights, so it is very important for a pruning method to effectively identify critical weights and prevent the neurons from failure. The effect of the threshold adaptation lies in two different aspects. The first one is to allow the network to reserve critical weights when the network is not

trained enough in the early phase of training. The second aspect is to balance the connection strength of excitatory neurons in the network so that more weights can be pruned from strong neurons and less from weak neurons to avoid causing substantial performance degradation of some neurons since neurons are critical processing units in the network. Moreover, a post-training

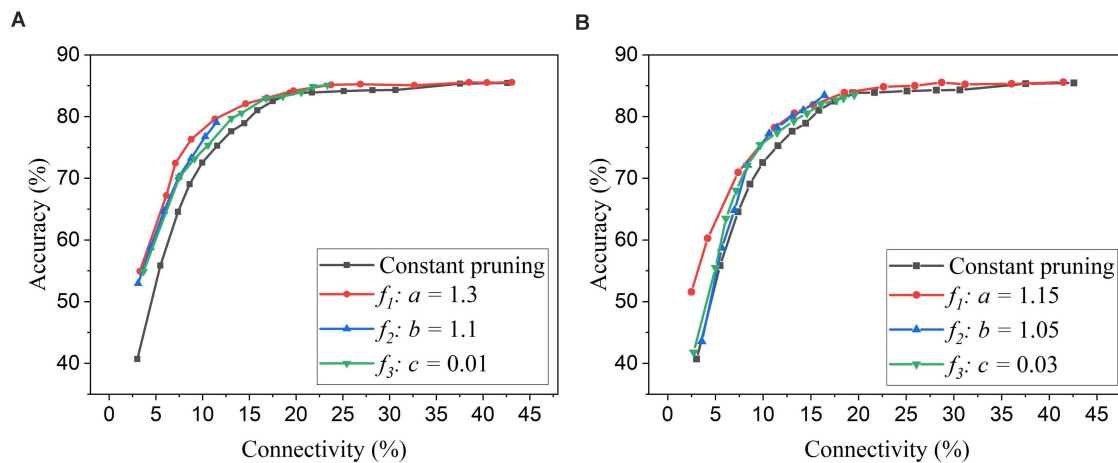


FIGURE 7 | Performance comparison among different adaptation functions in the SNN trained on MNIST dataset with 100 neurons. **(A)** APT: Online adaptive pruning over time, and **(B)** APN: Online adaptive pruning over neurons. The spike count interval is set as 30.

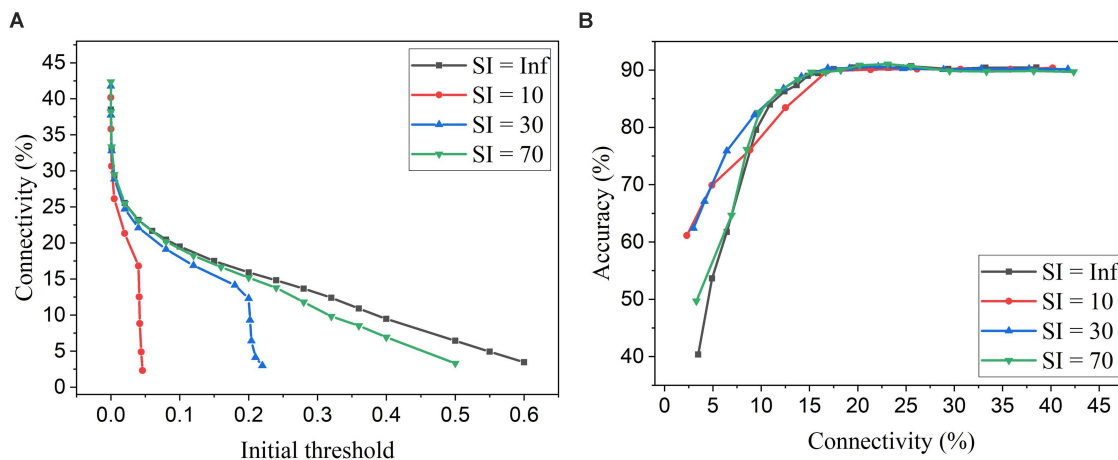


FIGURE 8 | Simulation results of online adaptive pruning over neurons for different spike count intervals (SI) in the SNN trained on MNIST dataset with 100 neurons. **(A)** Connectivity vs. initial threshold, and **(B)** Accuracy vs. connectivity. SI = Infinity (Inf) means that there is only one group and hence no adaptation over neurons.

pruning method is included for comparison (Rathi et al., 2019). Instead of pruning weight while training, this method prunes weights after the training process is done. It shows slightly better performance than the online constant pruning method but much worse performance than the proposed online APTN method when the connectivity is smaller than 10%. This is because, during the online constant pruning process, some critical weights can be mistakenly removed, whereas the adaptive method can effectively reserve the critical weights and provide more chances for them to be trained. The proposed pruning methods were also studied in the SNN with 800 excitatory neurons trained on both datasets. The adaptation function f_1 was used. The time adaptation factor, the neurons adaptation factor, and the spike count interval were optimized and selected as 1.2, 1.15, and 30, respectively. The comparison among the proposed pruning methods is shown in **Figures 9B,D**. The pruning results show similar comparisons, and the same analysis can be applied. The post-training pruning

shows slightly better performance than the online constant pruning method but worse performance than the proposed online adaptive pruning methods. The pruning results further confirm that the proposed APTN method outperforms the other weight pruning methods, especially when the network becomes very sparse (connectivity < 10%). We can draw a conclusion that the proposed APTN method is the most effective pruning method that can significantly reduce network connectivity and maintain high accuracy.

For the proposed online pruning method, it is crucial to find the right starting point for pruning during training. If pruning starts too early, weights are not learned enough, and hence some critical weights can be mistakenly removed. While the weights are learned for enough time after 30,000 training images, the remaining training process will further fine-tune the unpruned critical weights as they get more chance for STDP updates when more weights are pruned. This is due to the dynamics of the

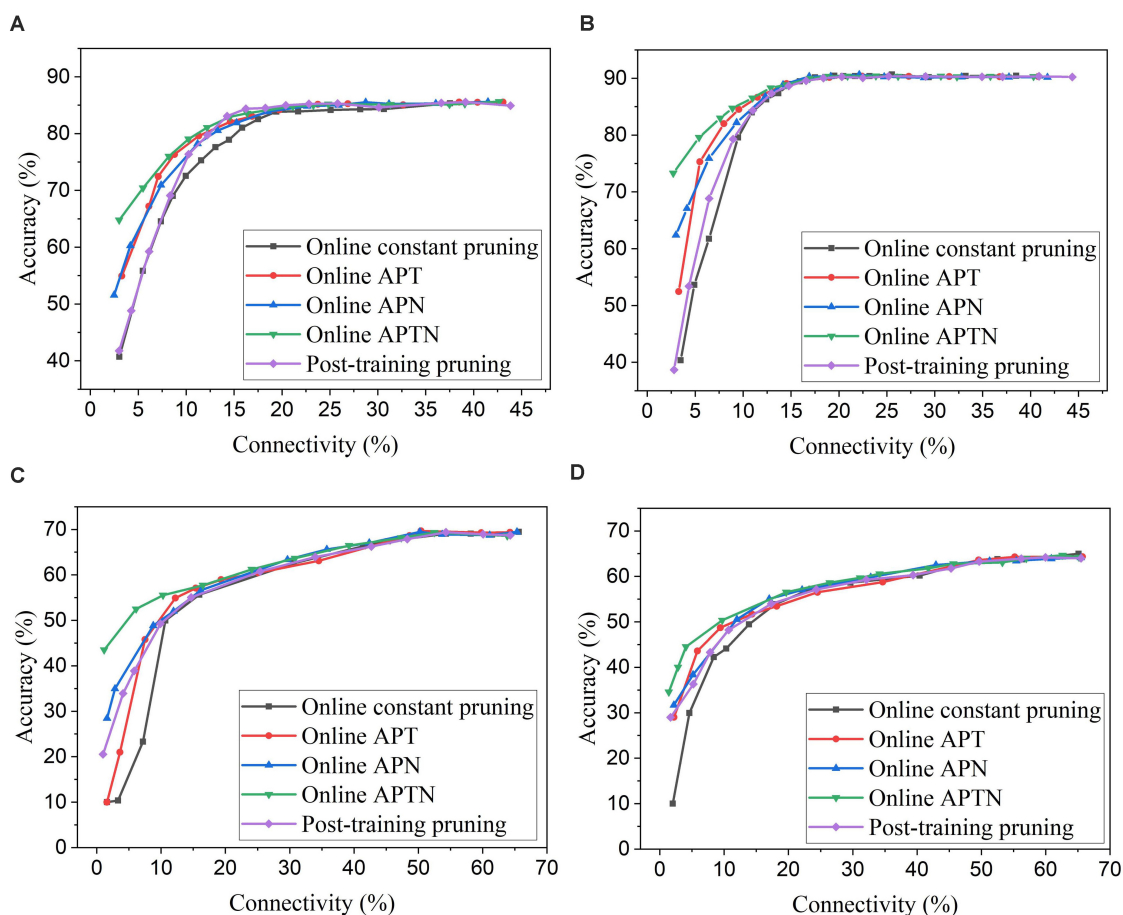


FIGURE 9 | Comparison among different weight pruning methods in the SNN trained on different datasets. MNIST dataset: **(A)** 100 excitatory neurons and **(B)** 800 excitatory neurons. Fashion-MNIST dataset: **(C)** 100 excitatory neurons and **(D)** 800 excitatory neurons.

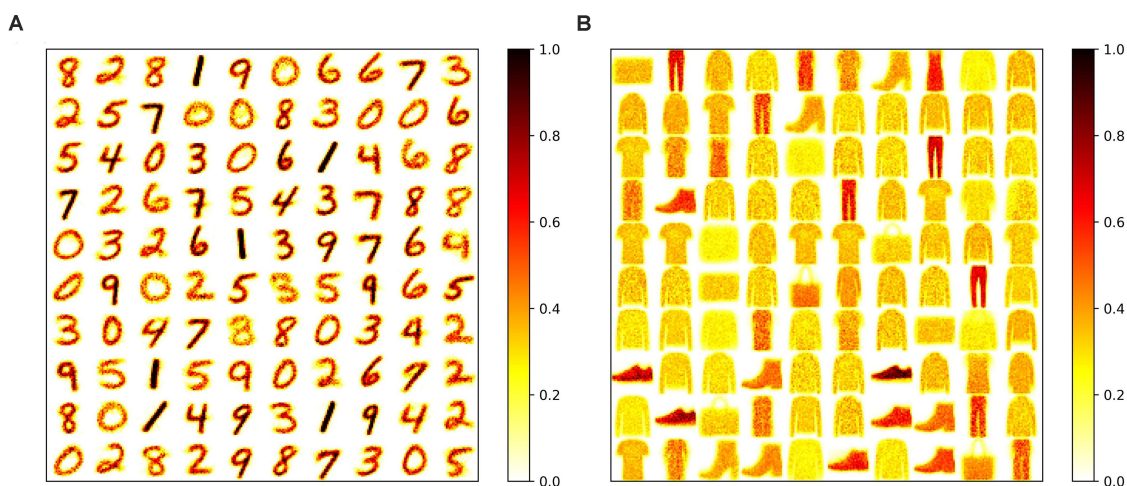
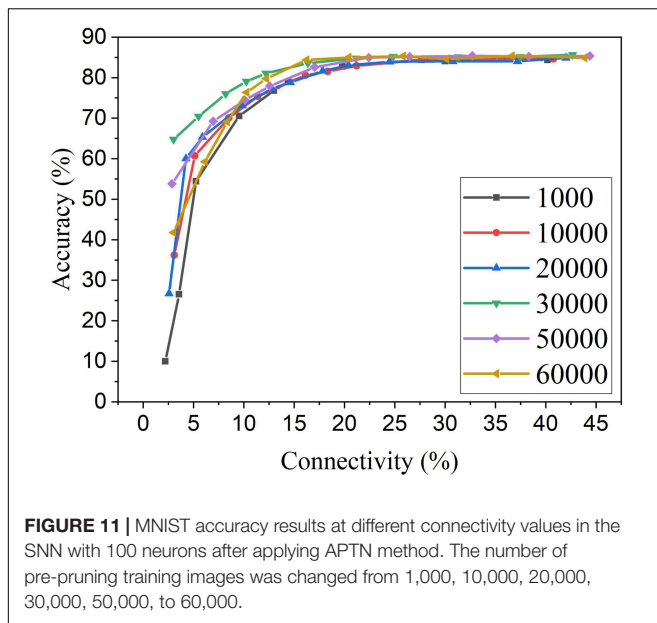


FIGURE 10 | Trained weight maps on **(A)** MNIST dataset and **(B)** Fashion-MNIST dataset in the SNN with 100 neurons without pruning. Each pattern in the maps is formed by arranging the weights associated with each neuron to a 28 × 28 matrix.



STDP learning rule that the weights with more contribution to the neural firing are strengthened more often. So if pruning starts during the last stage of training, the unpruned weights will not have enough chance to be fine-tuned to preserve good network performance. The adaptation will not be carried out effectively, and there will be less improvement in accuracy and training energy efficiency. In section “Methods and results,” we decided to start pruning after training over 30,000 images based on the change of network dynamics. To further investigate the impact of the number of pre-pruning training images, we have obtained pruning results for the various number of pre-pruning training images which are presented in **Figure 11**. It confirms that pruning too early causes more performance loss. If the pruning happens at the later stage of training (50,000 pre-pruning training images), the performance loss is also observed, as there is almost no adaptation effect. 30,000 is proven to be the optimal point where neurons start to fire stably, and weight updates start to stabilize. Different from the post-training pruning method (60,000 pre-pruning training images), the online APTN method requires a crucial starting point during training in order to achieve the best network performance, and it also provides more chance for the unpruned critical weights to be trained during the training process. Starting at the 30,000 point, the online APTN method outperforms the post-training pruning method, especially when the network becomes very sparse, as demonstrated before.

Additionally, the selection of an adaptation function and the corresponding adaptation factors can be further optimized with more choices of functions and a finer grid of factor values. However, this is not in the scope of this work that aims to demonstrate the effectiveness of the proposed adaptive pruning method.

Computational Cost Reduction

In general, for neuromorphic hardware systems, like TrueNorth, SpiNNaker, and Loihi, the fundamental operation is the synaptic

event that occurs when a spike is transmitted from a source neuron to a target neuron. So the computational energy of an SNN is proportional to the synaptic activity (Merolla et al., 2014). Pruning leads to a reduced number of synapses in the network and hence less synaptic events. To evaluate the energy improvement benefit of our proposed adaptive pruning method, we computed the number of SOPs per image (SOPs/image) during both training and inference. The training SOPs include weight accumulations and STDP updates, while the inference SOPs only count weight accumulations. The results for the SNNs trained on MNIST dataset with 100 and 800 excitatory neurons are shown in **Figure 12A**. The SOPs/image is normalized to the value obtained from the SNN without pruning. Clearly, the SOPs/image during both training and inference decreases almost linearly with connectivity, as the number of synaptic events is proportional to the number of unpruned synapses. The inference SOPs/image is reduced more significantly than the training SOPs/image. Moreover, the online pruning method can effectively reduce the number of training SOPs and hence improve training energy efficiency, making it promising for improving online learning systems. To help choose the network connectivity to reach the best overall performance, we define a figure of merit by considering accuracy loss and the total SOPs/image (training + inference) as below.

$$FOM = Accuracy\ loss \times Normalized\ total\ SOPs/image$$

The defined FOM is used on a per-network basis to help identify the best network connectivity for that specific network, as demonstrated in **Figure 12B**. As a result, the best choices of the connectivity are 14.5% and 17% for 100-neuron and 800-neuron networks, respectively. Specifically, at 14.5% connectivity, the adaptive pruning method leads to a 27% reduction in SOPs/image during training and a 60% reduction during inference with 2.85% accuracy loss in the SNN with 100 excitatory neurons. In the case of 800 excitatory neurons, at 17% connectivity, the method leads to a 30% reduction during training and a 55% reduction during inference with only 0.44% accuracy loss. It should be noted that the proposed FOM provides one way to determine the best network connectivity, and other factors or definitions could also be applied depending on the requirements of specific applications.

Comparison With Prior Works

Neuron pruning is one of the structured weight pruning strategies, which eliminates all the weights associated with the pruned neurons and reduces the network complexity proportionally. However, directly removing neurons from the network could cause severe deterioration of network performance. We compared the proposed online weight pruning methods with an online adaptive neuron pruning method presented in our previous work (Guo et al., 2020). The comparison is shown in **Figure 13**. In **Figure 13A**, the online adaptive neuron pruning method shows worse accuracy than the weight pruning methods, which proves that weight pruning is more effective in preserving network performance. Despite the severe accuracy drop, the neuron pruning method requires

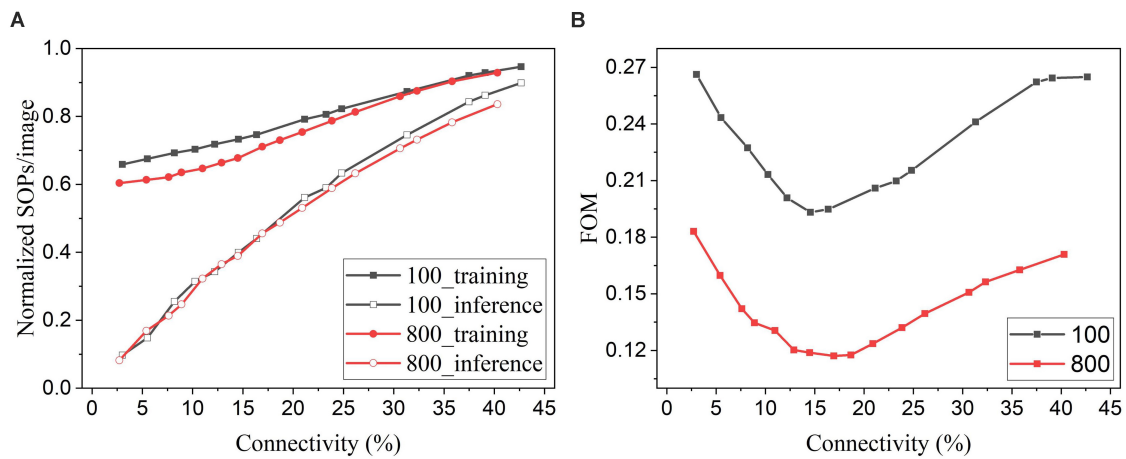


FIGURE 12 | (A) Normalized SOPs/image and **(B)** a figure of merit (FOM) for different connectivity values are obtained in the SNNs with 100 and 800 excitatory neurons using the online adaptive pruning over time and neuron method.

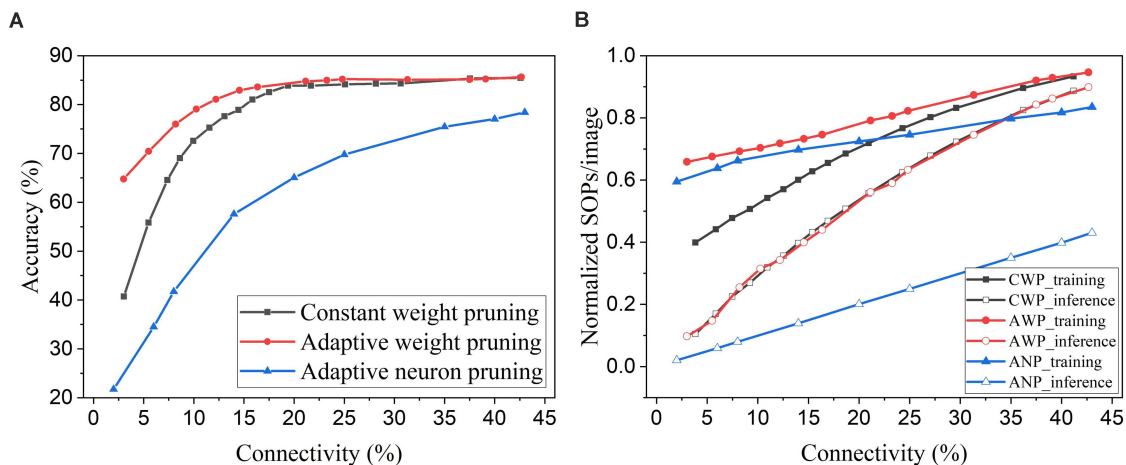


FIGURE 13 | Comparison between online weight pruning methods and an online adaptive neuron pruning method in the SNN trained on MNIST dataset with 100 excitatory neurons. **(A)** Accuracy and **(B)** normalized SOPs/image change with connectivity. CWP, AWP, and ANP are short for constant weight pruning, adaptive weight pruning, and adaptive neuron pruning, respectively.

fewer training SOPs/image than the adaptive weight pruning method and can reduce the inference SOPs/image much more significantly. Moreover, an additional benefit of the neuron pruning method is the elimination of state memory and processing power of the pruned neurons.

An online soft weight pruning method for unsupervised SNNs was reported in Shi et al. (2019). Unlike conventional pruning methods, instead of removing the pruned weights, this method sets the pruned weights constant at the lowest possible weight value or the current value and stops updating them for the rest of the training process. By setting the pruned weights to the lowest possible value, the soft pruning method is equivalent to the constant pruning method in our case since the lowest value is 0. In this comparison, we refer to the soft pruning method as the case where the pruned weights are kept constant at their current values. Since the soft pruning method does not induce the

sparsity in the network, the connectivity remains 100% and hence is not applicable in the comparison. Instead, we use the unpruned percentage that is the percentage of the unpruned weights in the total weights before pruning. In **Figure 14A**, it can be seen that the soft pruning method starts to have performance improvement over the constant pruning method after the unpruned percentage drops below 10%. Our proposed adaptive pruning method gives better performance when the unpruned percentage is between 5% and 20%, but worse performance after the unpruned percentage drops below 5%. When most of the weights are pruned, the soft pruning method is still able to retain high accuracy by keeping the pruned weights that were trained for some time in the network. However, the soft pruning method brings less benefit to the computational cost compared with the adaptive pruning method. **Figure 14B** shows that it contributes to less reduction in training SOPs/image and no reduction in inference

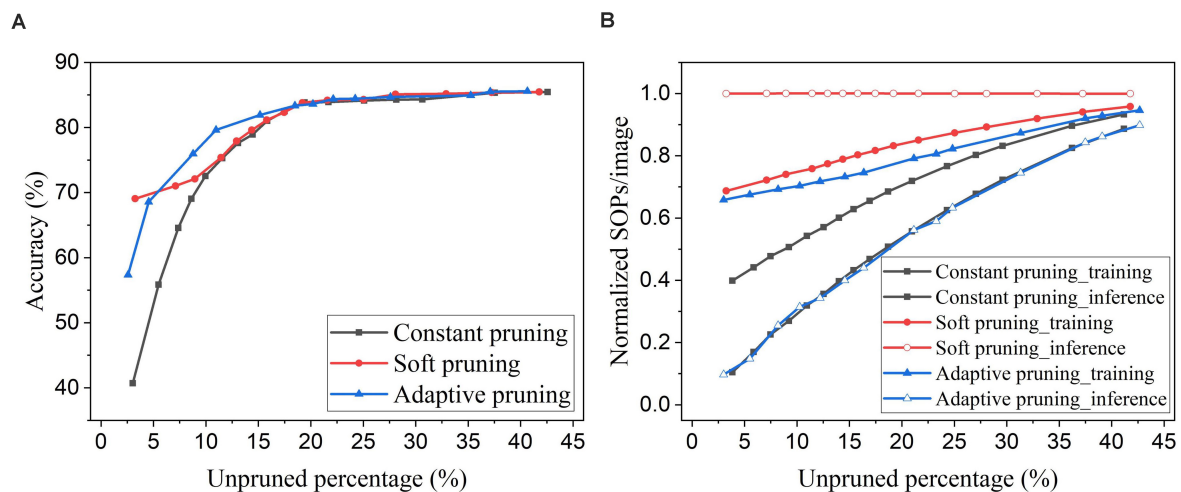


FIGURE 14 | Comparison with the online soft weight pruning method adopted from Shi et al. (2019) in the SNN trained on MNIST dataset with 100 excitatory neurons. **(A)** Accuracy and **(B)** normalized SOPs/image change with unpruned weights percentage. Since the soft pruning method does not remove the pruned weights, the connectivity is not applicable as the x axis here. Instead, the unpruned percentage is used, which is defined as the percentage of the unpruned weights in the total weights before pruning.

TABLE 2 | Comparison among different pruning methods in the SNN trained on MNIST dataset.

Pruning methods	Accuracy loss 100/800	Training SOPs reduction 100/800	Inference SOPs reduction 100/800
Online adaptive neuron pruning Guo et al., 2020	38.64%/38.85%	33%/48%	90%/90%
Post-training weight pruning Rathi et al., 2019	9.43%/9.21%	0%/0%	70%/71%
Online soft weight pruning Shi et al., 2019	12.45%/5.94%	25%/24%	0%/0%
Online constant weight pruning Shi et al., 2019	13.23%/6.73%	46%/46%	70%/69%
Online adaptive weight pruning (Our work)	6.73%/3.87%	30%/36%	69%/68%

Accuracy loss and SOPs reduction for two network sizes (100 and 800 neurons) are shown. The connectivity is selected as 10%.

SOPs/image. In comparison, our proposed adaptive pruning method can lead to more reduction in SOPs/image, especially during inference. The constant pruning method gives the most improvement in decreasing the training SOPs/image when a large number of weights are pruned at the cost of severe accuracy loss, because it applies a large constant threshold throughout the whole pruning process.

Comprehensive comparisons among different pruning methods in terms of accuracy loss and SOPs are provided in **Algorithm 1**, **Table 3**, including results for two network sizes and two datasets. The reduction is defined as the reduced percentage of the SOPs/image by pruning against the SOPs/image in the SNN without pruning. Network connectivity is selected as 10%. The neuron pruning method achieves the highest reduction in inference SOPs but the worst accuracy loss on both datasets. The post-training weight pruning method is able to produce small accuracy loss but no reduction in training SOPs. The soft online pruning method leads to the least accuracy loss on Fashion-MNIST dataset, because classifying more complex patterns in the dataset is more sensitive to the weights loss and this pruning method keeps the pruned weights in the network at their current values instead of removing them. However, this method leads to no benefits in reducing inference operations. The constant online pruning method can reduce both training

and inference operations effectively at the cost of high accuracy loss. Our method achieves the least accuracy loss on MNIST dataset and slightly higher accuracy loss on Fashion-MNIST dataset than the soft online pruning method. Our method can lead to a large reduction in SOPs comparable to the constant online weight pruning and adaptive online neuron pruning methods during both training and inference. The network size has no substantial impact on the comparisons. In conclusion, our proposed adaptive pruning method can significantly reduce computational operations during both training and inference and maintain high accuracy at the same time.

Implementation Overhead

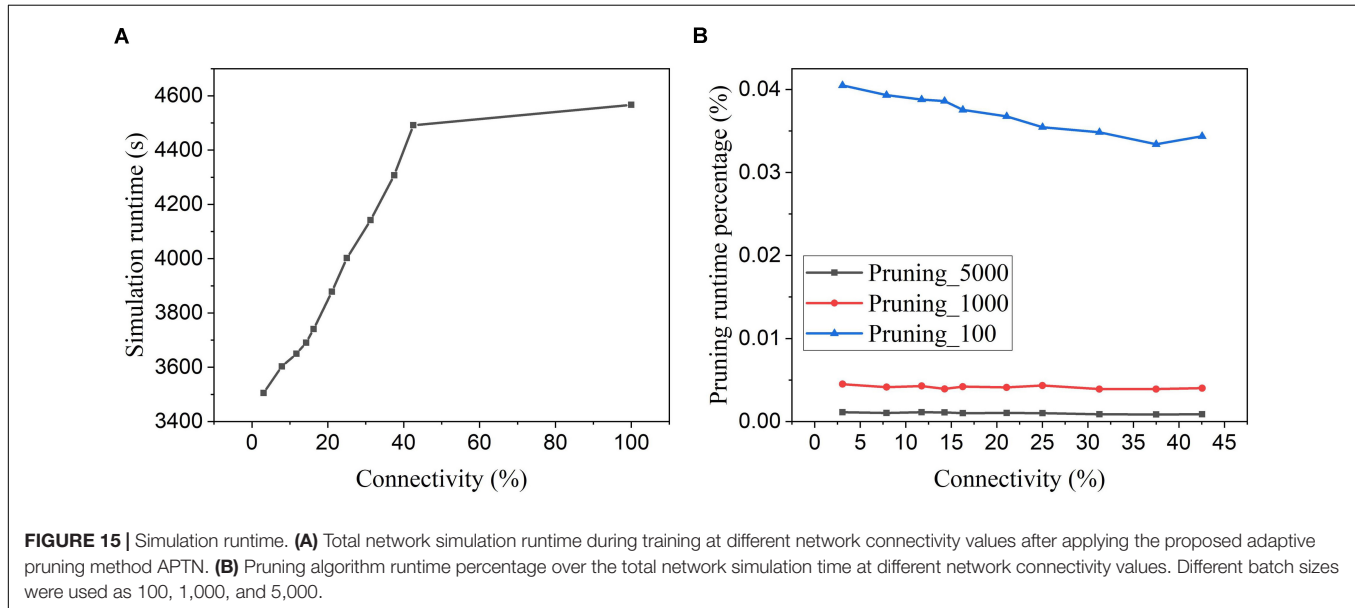
The proposed adaptive pruning algorithm can be implemented in hardware systems without adding significant overhead. To investigate the overhead, we chose three metrics: processing speed, area, and energy.

Figure 15A shows the software simulation runtime of the whole network during training, including the time used for executing the pruning algorithm. The software simulation is programmed in Python language and runs sequentially in a single process. The runtime decreases with the increasing pruning percentage (decreasing connectivity), which proves that the proposed online pruning method is able to shorten the network

TABLE 3 | Comparison among different pruning methods in the SNN trained on Fashion-MNIST dataset.

Pruning methods	Accuracy loss 100/800	Training SOPs reduction 100/800	Inference SOPs reduction 100/800
Online adaptive neuron pruning Guo et al., 2020	42.14%/40.88%	23%/39%	90%/90%
Post-training weight pruning Rath et al., 2019	16.35%/19.98%	0%/0%	82%/85%
Online soft weight pruning Shi et al., 2019	12.23%/9.63%	27%/29%	0%/0%
Online constant weight pruning Shi et al., 2019	20.43%/19.24%	45%/49%	88%/87%
Online adaptive weight pruning (Our work)	14.09%/13.67%	28%/31%	85%/84%

Accuracy loss and SOPs reduction for two network sizes (100 and 800 neurons) are shown. The connectivity is selected as 10%.

**TABLE 4** | Estimated number of clock cycles and computational operations (Ops) for the pruning algorithm and SNN training phase in the network with 100 neurons.

Phase	Pruning (single batch)			Pruning (average per image) Batch: 100/5,000	SNN Training (average per image)
	Grouping	Adapting	Weight pruning		
# Cycles	120	2,000	78,400	404/9	> 96,099
# Ops	400	1,100	156,800	794/18	96,099

Two batch sizes (100 and 5,000) were used for estimating the average per image. The number of operations for SNN training only includes synaptic operations obtained at the connectivity of 10%.

runtime as it reduces the number of SOPs, including weight accumulations and STDP updates. Besides, the APTN pruning runtime is negligible compared to the total runtime (SNN runtime plus pruning runtime). For example, in **Figure 15B**, the pruning runtime percentage is around 0.001% at the batch size of 5,000 and less than 0.04% even when the batch size is decreased to 100. For hardware runtime, we estimated the number of clock cycles required to run the pruning algorithm in a general synchronous digital system, as shown in **Table 4**. At each batch, the proposed pruning process requires three essential phases, including dividing the neuron groups (grouping phase), adapting pruning thresholds over neurons (adapting phase), and writing 0s to weight memory (weight pruning phase) operations. The grouping method with sorting in the proposed algorithm can be replaced by simply searching for the minimum and maximum values of firing activities of neurons and dividing the

whole range of firing activity (max – min) according to the spike interval without performance loss. The adapting phase is simply to position each neuron in the right group according to its firing activity and assign the corresponding pruning threshold. Both grouping and adapting phases depend on the number of groups that varies over time but is smaller than 20. We used 20 for the estimation. For both phases, we assume that no parallelism is applied for estimating the upper limit. Moreover, we assume that all the weights are stored in one memory, and the weight pruning operations can only access one weight at a time. However, it should be noted that multiple accesses to weight memory are available in practice. So the estimation is at the upper limit of the running cycles of the pruning algorithm. The estimated number of different phases in the table is for single-batch pruning. The number of clock cycles for the SNN training phase is much larger than the number of training SOPs/image, 96,099, since

TABLE 5 | Estimated number of essential digital gates and memories required for the pruning algorithm and equivalent NAND gates.

Pruning unit					SNN	
Sub/Add (16 bits)	Comparator (16 bits)	Register	NAND	BRAM (18 Kb)	NAND	BRAM (18 Kb)
4	18	620	8284	2	3.0×10^6	50

The number of NAND gates and BRAMs for an SNN were obtained with 100 neurons according to the proposed digital implementation from (Guo et al., 2020).

each SOP includes many processes, such as searching for destination addresses, reading out synaptic weights, routing spikes to the destination, and weight addition or STDP update, which takes multiple cycles to finish. It can be seen that the average number of clock cycles per image for pruning with a small batch size of 100 is far smaller than the number of training SOPs. Therefore, the hardware runtime of the pruning algorithm is negligible.

For energy overhead, the number of basic operations, such as addition, comparison, and memory access, was estimated in **Table 4** for different pruning phases. For the estimation, 16 bits and 8 bits were used to represent the integer part and fractional part, respectively. The multiplication operation involved in the algorithm can be approximated by shift and addition operations. The grouping phase requires addition, comparison, and memory access operations, while the adapting phase only needs comparison and memory access operations. The operations in the weight pruning phase involve memory access and comparison between weights and a pruning threshold. The average number of operations per image is 794 and 18 for the batch size of 100 and 5,000, respectively, which are very small compared to the number of training SOPs/image. For energy comparison, we take an example of SNN implementation on Loihi neuromorphic hardware (Davies et al., 2018). The reported minimum energy/SOP on this hardware is 23.6 pJ. So the minimum SOP energy per image is around 2.3 uJ. Since memory access consumes more energy than addition and comparison operations, we used the energy of memory access for all the operations for the comparison. The memory access (read and write) to an SRAM cell under the same technology consumes around 0.5 pJ (Yang et al., 2016). So, the estimated energy for pruning operations per image is 3.4 nJ at the batch size of 100, which is around 0.1% of the SOP energy. Besides, the network also spends energy on updating neural states in neural cores, which makes the percentage even smaller. Thus, we can claim that the energy overhead is negligible.

As for area overhead, the number of essential digital gates and memories required to implement the pruning algorithm and equivalent NAND gates was estimated in **Table 5**. Each weight needs a flag bit to indicate if it has been pruned. This bit can be simply attached to the weight bits in the memory with very little overhead. The number of NAND gates for an SNN with 100 neurons was estimated according to the proposed digital implementation from Guo et al. (2020). Clearly, the number of equivalent NAND gates for the pruning algorithm is much smaller than that for the SNN. For example, the number of equivalent NAND gates in the pruning unit is only around 0.3% of that in the SNN. For memory comparison, in the pruning unit, firing activity and pruning threshold of neurons are assumed to

be stored in block RAMs (BRAMs). Two 18 Kb BRAMs are totally enough, which is much smaller than the memory size required in the SNN. Therefore, the area overhead is very small.

Impact and Future Work

The proposed adaptive method would be effective in improving the compression rate and preserving good network performance in other neural networks, as different threshold adaptation techniques have also been applied to improve the pruning performance in other neural networks.

The iterative pruning method has been the most successful and popular pruning technique in ANNs, which relies on numerous cycles of training and pruning in order to induce sparsity in weight matrices and preserve network performance (Han et al., 2015). This method iteratively sets the weights below a certain threshold to zero and retrains the network to regain its performance. The main limitation is the need to manually tweak the thresholds for neurons in different layers to achieve the best results by iterative tuning. While this iterative method can effectively compress networks, it requires a large amount of time and resources in order to find the optimized sparse networks, which hinders its use in large-scale applications.

In order to eliminate the need for iterative threshold tuning, many works have explored to adapt threshold values for neurons in different layers by training the thresholds together with weights (Manessi et al., 2018; Ye et al., 2019; Azarian et al., 2020). These methods use the same concept of adapting threshold spatially as in our method based on the fact that neurons in different layers have different sensitivity to pruning thresholds, but in a different adaptation process. In our method, we used the firing activity of neurons to determine their pruning thresholds, while these methods adapt the thresholds based on the network loss in a supervised fashion. These methods were able to find the optimal thresholds for each layer and do not require pruning-retraining cycles. The results have shown that with the threshold adaptation, their methods can achieve a much larger compression rate with higher classification accuracy than the method without adaptation. Moreover, threshold adaptation over time during training was demonstrated to be beneficial in accelerating the pruning process and achieving a higher compression rate. Narang et al. (2017) proposed to adapt the pruning threshold over time using a monotonically increasing function during training. A heuristic function was presented to calculate the threshold at different iteration steps, which requires many hyperparameters. They tested the method in different types of recurrent neural networks (RNNs) and demonstrated that this adaptive method could achieve better network performance and a higher compression rate without pruning-retraining cycles than a hard pruning method that simply prunes the weights with a constant

threshold. Therefore, we believe that the proposed adaptive pruning method can be useful in improving the compression rate and preserving good network performance in ANNs. To test the versatility of our method, we will investigate the impact of the proposed adaptive method in deep SNNs in our future works.

CONCLUSION

In this work, we proposed an online adaptive weight pruning method that adapts the pruning threshold over time and neurons during training in an unsupervised SNN. The effects of the threshold adaptation over time and neurons were studied individually. Different functions used to adapt the threshold were applied and compared. It is demonstrated that both adaptation over time and neurons can improve the network performance against an online constant weight pruning method. The adaptation enables the network to reserve critical weights when the network is not trained enough at the early phase of training and balance the connection strength of excitatory neurons in the network to avoid largely deteriorating the performance of weak neurons. So, combining the two adaptation schemes can further improve network performance. The online adaptive pruning method provides better performance than the post-training pruning method, suggesting that it can not only improve training energy efficiency but also achieve higher accuracy. Regarding the computational cost, the number of SOPs was analyzed, which shows that the proposed online adaptive pruning method can significantly reduce the SOPs/image during both training and inference. Furthermore, comparisons with the previous works reveal that our method can lead to better accuracy and a more significant reduction in SOPs. The implementation overhead of the proposed method was evaluated in terms

of processing speed, area, and energy, which is proven to be negligible in the network. Therefore, the proposed online adaptive pruning method provides a promising approach for reducing network complexity and improving energy efficiency with good performance in SNNs for real-time applications.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

WG, MF, HY, AE, and KS: conceptualization. WG, MF, HY, AE, and KS: methodology. WG: software, algorithms and writing – original draft preparation. WG, MF, and HY: investigation and validation. MF, HY, AE, and KS: writing – review and editing. AE and KS: supervision. KS: project administration. All authors contributed to the article and approved the submitted version.

FUNDING

This research was funded by King Abdullah University of Science and Technology (KAUST) AI Initiative.

ACKNOWLEDGMENTS

We acknowledge the financial support from King Abdullah University of Science and Technology (KAUST), Saudi Arabia.

REFERENCES

- Anwar, S., Hwang, K., and Sung, W. (2017). Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.* 13:32. doi: 10.1145/3005348
- Azarian, K., Bhalgat, Y., Lee, J., and Blankevoort, T. (2020). Learned threshold pruning. *ArXiv [Preprint]*. Available online at: <https://arxiv.org/pdf/2003.00075.pdf> (accessed October 4, 2020).
- Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: i. homogeneous synaptic input. *Biol. Cybernet.* 95, 1–19. doi: 10.1007/s00422-006-0068-6
- 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
- Diehl, P., 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
- Fontaine, B., Peña, J. L., and Brette, R. (2014). Spike-threshold adaptation predicted by membrane potential dynamics in vivo. *PLoS Comput. Biol.* 10:e1003560. doi: 10.1371/journal.pcbi.1003560
- 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
- Guo, W., Yantur, H. E., Fouda, M. E., Eltawil, A. M., and Salama, K. N. (2020). Towards efficient neuromorphic hardware: unsupervised adaptive neuron pruning. *Electronics* 9:1059.
- Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). “Learning both weights and connections for efficient neural networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems – Volume 1*, (Montreal: MIT Press).
- Iglesias, J., Eriksson, J., Grize, F., Tomassini, M., and Villa, A. E. P. (2005). Dynamics of pruning in simulated large-scale spiking neural networks. *Biosystems* 79, 11–20. doi: 10.1016/j.biosystems.2004.09.016
- 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
- Li, H., Liu, N., Ma, X., Lin, S., Ye, S., Zhang, T., et al. (2019). “ADMM-based weight pruning for real-time deep learning acceleration on mobile devices,” in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, (Tysons Corner, VA: Association for Computing Machinery).
- Manessi, F., Rozza, A., Bianco, S., Napolitano, P., and Schettini, R. (2018). “Automated Pruning for Deep Neural Network Compression,” in *Proceedings of the 2018 24th International Conference on Pattern Recognition (ICPR)*, Beijing, 657–664.
- Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356
- 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. doi: 10.1126/science.1254642
- Narang, S., Diamos, G., Sengupta, S., and Elsen, E. (2017). Exploring sparsity in recurrent neural networks. *ICLR*

- 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
- Paupamah, K., James, S., and Klein, R. (2020). "Quantisation and pruning for neural network compression and regularisation," in *Proceedings of the 2020 International SAUPEC/RobMech/PRASA Conference*, Cape Town, 1–6. doi: 10.1109/SAUPEC/RobMech/PRASA48453.2020.9041096
- Pfister, J.-P., and Gerstner, W. (2006). Triplets of spikes in a model of spike timing-dependent plasticity. *J. Neurosci.* 26, 9673–9682. doi: 10.1523/jneurosci.1425-06.2006
- Rathi, N., Panda, P., and Roy, K. (2019). STDP-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition. *IEEE Trans. Comput. Aided Des. Integr. Circ. Syst.* 38, 668–677. doi: 10.1109/TCAD.2018.2819366
- Shi, Y., Nguyen, L., Oh, S., Liu, X., and Kuzum, D. (2019). A soft-pruning method applied during training of spiking neural networks for in-memory computing applications. *Front. Neurosci.* 13:405. doi: 10.3389/fnins.2019.00405
- Shrestha, A., and Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access* 7, 53040–53065. doi: 10.1109/ACCESS.2019.2912200
- Sredojevic, R., Cheng, S., Supic, L., Naous, R., and Stojanovic, V. (2017). Structured deep neural network pruning via matrix pivoting. *ArXiv [Preprint]*. Available online at: <https://arxiv.org/abs/1712.01084#:~:text=In%20this%20work%20we%20introduce,for%20obtaining%20resource%20efficient%20DNNs> (accessed July 15, 2020).
- 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
- Tung, F., and Mori, G. (2020). Deep neural network compression by in-parallel pruning-quantization. *IEEE Trans. Pattern Anal. Mach. Intellig.* 42, 568–579. doi: 10.1109/TPAMI.2018.2886192
- Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *ArXiv [Preprint]*. Available online at: <https://arxiv.org/abs/1708.07747> (accessed October 11, 2020).
- Yang, Y., Jeong, H., Song, S. C., Wang, J., Yeap, G., and Jung, S. (2016). Single Bit-Line 7T SRAM cell for near-threshold voltage operation with enhanced performance and energy in 14 nm FinFET technology. *IEEE Trans. Circ. Syst. I Regul. Pap.* 63, 1023–1032. doi: 10.1109/TCSI.2016.2556118
- Ye, S., Feng, X., Zhang, T., Ma, X., Lin, S., Li, Z., et al. (2019). Progressive DNN compression: a key to achieve ultra-high weight pruning and quantization rates using ADMM. *ArXiv [Preprint]*. Available online at: <https://arxiv.org/abs/1903.09769> (accessed October 4, 2020).
- Zillmer, E. A., and Spiers, M. V. (2001). *Principles of Neuropsychology*. Belmont, CA: Wadsworth/Thomson Learning.

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 Guo, Fouda, Yantir, Eltawil and Salama. 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 Spike-Driven Learning With Dendritic Event-Based Processing

Shuangming Yang¹, Tian Gao¹, Jiang Wang¹, Bin Deng^{1*}, Benjamin Lansdell² and Bernabe Linares-Barranco³

¹ School of Electrical and Information Engineering, Tianjin University, Tianjin, China, ² Department of Bioengineering, University of Pennsylvania, Philadelphia, PA, United States, ³ Microelectronics Institute of Seville, Seville, Spain

OPEN ACCESS

Edited by:

Angelo Arleo,
Centre National de la Recherche
Scientifique (CNRS), France

Reviewed by:

Charles Augustine,
Intel, United States
Sio Hoi Ieng,
Université Pierre et Marie Curie,
France

*Correspondence:

Bin Deng
dengbin@tju.edu.cn

Specialty section:

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

Received: 31 August 2020

Accepted: 21 January 2021

Published: 19 February 2021

Citation:

Yang S, Gao T, Wang J, Deng B,
Lansdell B and Linares-Barranco B
(2021) Efficient Spike-Driven Learning
With Dendritic Event-Based
Processing.
Front. Neurosci. 15:601109.
doi: 10.3389/fnins.2021.601109

A critical challenge in neuromorphic computing is to present computationally efficient algorithms of learning. When implementing gradient-based learning, error information must be routed through the network, such that each neuron knows its contribution to output, and thus how to adjust its weight. This is known as the credit assignment problem. Exactly implementing a solution like backpropagation involves weight sharing, which requires additional bandwidth and computations in a neuromorphic system. Instead, models of learning from neuroscience can provide inspiration for how to communicate error information efficiently, without weight sharing. Here we present a novel dendritic event-based processing (DEP) algorithm, using a two-compartment leaky integrate-and-fire neuron with partially segregated dendrites that effectively solves the credit assignment problem. In order to optimize the proposed algorithm, a dynamic fixed-point representation method and piecewise linear approximation approach are presented, while the synaptic events are binarized during learning. The presented optimization makes the proposed DEP algorithm very suitable for implementation in digital or mixed-signal neuromorphic hardware. The experimental results show that spiking representations can rapidly learn, achieving high performance by using the proposed DEP algorithm. We find the learning capability is affected by the degree of dendritic segregation, and the form of synaptic feedback connections. This study provides a bridge between the biological learning and neuromorphic learning, and is meaningful for the real-time applications in the field of artificial intelligence.

Keywords: spiking neural network, credit assignment, dendritic learning, neuromorphic, spike-driven learning

INTRODUCTION

Learning requires assigning credit to each neuron for its contribution to the final output (Bengio et al., 2015; Lillicrap et al., 2016). How a neuron determines its contribution is known as the credit assignment problem. In particular, the training of deep neural networks is based on error back-propagation, which uses a feedback pathway to transmit information to calculate error signals in the hidden layers. However, neurophysiological studies demonstrate that the conventional error

back-propagation algorithm is not biologically plausible. One problem is known as weight transport: backpropagation utilizes a feedback structure with the exact same weights as the feed-forward pathway to communicate gradients (Liao et al., 2016). This symmetric feedback structure has not been proven to exist in biological neural circuit. Several studies have presented solutions to modify or approximate the backpropagation algorithm in a more biologically plausible manner (Lee et al., 2015; Scellier and Bengio, 2017; Lansdell and Kording, 2019; Lansdell et al., 2019). In fact, active channels in dendrites can drive different forms of spiking activities (Schmolesky et al., 2002; Larkum, 2013). A potential solution is thus to segregate signals into dendritic compartments, so that the credit signals can be kept separate from other ongoing computation (Richards and Lillicrap, 2019). Recent work shows how spiking neural networks can implement feedback structures that allow efficient solving of the credit assignment problem by dendritic computation (Urbanczik and Senn, 2014; Wilmes et al., 2016; Bono and Clopath, 2017; Guerguiev et al., 2017). Further, other work has shown that even feedback systems that crudely approximate the true feedback weights can solve some learning tasks (Zenke and Ganguli, 2018; Lee et al., 2020). Together these works show that the credit assignment problem can be largely solved by biologically plausible neural systems.

An ongoing challenge in neuromorphic computing is to present general and computationally efficient algorithms of deep learning. Previous works have shown how neuromorphic approaches for deep learning can be more efficient compared to Von Neumann architecture (Esser et al., 2015; Indiveri et al., 2015; Neftci et al., 2017). However, these systems have yet to be fully realized. By design, learning in neuromorphic hardware operates under similar constraints to learning in biological neural networks. The credit assignment problem, and the problem of weight transport also manifest in this setting: neuromorphic learning systems that do not require weight transport enjoy less data transfer between components. In this way, biologically plausible approaches to deep learning can also be used to make neuromorphic computing more efficient. Previous neuromorphic systems have been presented for high-performance brain-inspired computation, providing tests of biological learning models and real-time applications (Qiao et al., 2015; Yang et al., 2015, 2018, 2020, 2021).

Recent proposals for solutions to the credit assignment problem have not been considered in neuromorphic computing. Here we present a novel dendritic event-based processing (DEP) algorithm to facilitate the efficient implementation of the credit-assignment task on neuromorphic hardware. The presented DEP algorithm is inspired by the primary sensory areas of the neocortex, providing the segregation of feed-forward and feedback information required to compute local error signals and to solve the credit assignment problem. In the DEP algorithm, a binarization method and a dynamic fixed-point solution are presented for the efficient implementation of deep learning. The paper is organized as follows: section “Introduction” describes the preliminaries of this study, including neuromorphic computing and the spiking neural network (SNN) model. Learning with stochastic gradient descent (SGD) in spiking neural networks is

introduced and explained in section “Materials and Methods.” Section “Results” presents the experimental results. And finally, the discussions and conclusions are proposed in sections “Discussion” and “Conclusion,” respectively.

MATERIALS AND METHODS

Learning With Dendrites in Event-Driven Manner

Learning needs neurons to receive signals to assign the credit for behavior. Since the behavioral impact in early network layers is based on downstream synaptic connections, credit assignment in multi-layer networks is challenging. Previous solutions in artificial intelligence use the backpropagation of error algorithm, but this is unrealistic in the neural systems. Rather than requiring weight transport, current biologically plausible solutions to the credit assignment problem use segregated feed-forward and feedback signals (Lee et al., 2015; Lillicrap et al., 2016). In fact, the cortico-cortical feedback signals to pyramidal neurons can transmit the necessary error information. These works show how the circuitry needed to integrate error information may exist within each neuron. The idea is that both feed-forward sensory information in the neocortex and the higher-order cortico-cortical feedback are received by different dendritic compartments, including basal and apical dendrites (Spratling, 2002). In a pyramidal cell, distal apical dendrites are distant from the soma, and communicate with the soma based on active propagation using the apical dendritic shaft, driven predominantly by voltage-gated calcium channels (Katharina et al., 2016). Further, there exist dynamics of plateau potentials that generate prolonged upswings in the membrane potential. These are based on the nonlinear dynamics of voltage-gated calcium current, and drive bursting at the soma (Larkum et al., 1999). The plateau potentials of the apical dendritic activities can induce learning in pyramidal neurons *in vivo* (Bittner, 2015).

Inspired by these phenomena, a previous study has proposed a learning algorithm with segregated dendrites (Guerguiev et al., 2017). Based on this work, an efficient learning algorithm for neuromorphic learning is presented in this study. The idea is that the basal dendritic compartment is coupled to the soma for processing bottom-up sensory information, and the apical dendritic compartment is used to process top-down feedback information to calculate credit assignment and induce learning using plateau potentials. The basic computing unit we use on the large-scale conductance-based spiking neural network (LaCSNN) system is based on the integrate-and-fire (IF) principle. As shown in **Figure 1**, the simple spiking behaviors of the IF neurons can be triggered by excitatory input spikes. The new state of the neural membrane potential with an input arriving is determined by the last updating time and the previous state. Thus, the event-driven neuron only updates when an input spike is received. Then the membrane potential decay after the last update is retroactively calculated and applied. The synaptic weight is then used to contribute to the resulting membrane potential. A spike event is emitted when the membrane potential exceeds a spike threshold, and then the neural activity is reset and

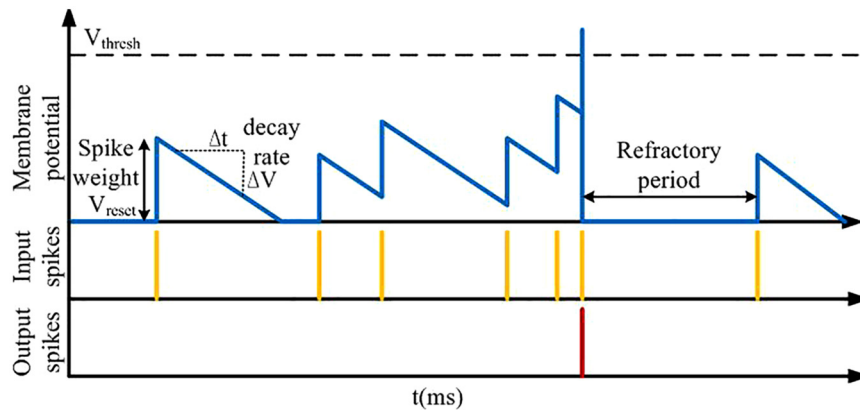


FIGURE 1 | Event-driven neural computing, showing the process of synaptic weights and time affecting the neural membrane potential and the refractory period.

mutual inhibition with coupled neurons is realized. Finally the membrane potential and spiking event are written to memory to store the network state of the next update of neural activity.

Network Architecture With SGD Algorithm

The network diagram utilizes the SNN model in the previous study by Guerguiev et al. (2017) as shown in **Figure 2**, which consists of an input layer with $m = 784$ neurons, a hidden layer with $n = 500$ neurons, and an output layer with $p = 10$ neurons. Since our primary interests are in the realization of neuromorphic networks, the proposed model is restricted to discrete systems based on the Euler method, where N is the time step for discretization. This way of representation is popular in the hardware implementation of spiking neural networks because of its feasibility of implementation and routing. Poisson spiking neurons are used in the input layer, whose firing rate is determined by the intensity of image pixels ranging from 0 to Φ_{max} . In the hidden layer, neurons are modeled using three functional compartments, which are basal dendrites, apical dendrites and soma. The membrane potential of the i th neuron in the hidden layer is updated as follows:

$$\tau \frac{V_i^0(N+1) - V_i^0(N)}{\Delta T} = -V_i^0(N) + \frac{g_b}{g_l} (V_i^{0b}(N) - V_i^0(N)) + \frac{g_a}{g_l} (V_i^{0a}(N) - V_i^0(N)) \quad (1)$$

where g_l , g_b , and g_a stand for the leak conductance, the basal dendrites conductance, and the apical dendrite conductance, and ΔT is the integration step. The superscript “0,” “a,” and “b” represent hidden layer, basal dendrite and apical dendrite. The parameter $\tau = C_m/g_l$ is a time constant, where C_m represents the membrane capacitance. The variables V^0 , V^{0a} , and V^{0b} represent the membrane potentials of soma, apical dendrite and basal dendrite, respectively. The dendritic compartments are defined

as weighted sums for the i th hidden layer neuron as follows:

$$\begin{cases} V_i^{0b}(N) = \sum_{j=1}^m W_{ij}^0 s_j^{input}(N) + b_i^0 \\ V_i^{0a}(N) = \sum_{j=1}^p Y_{ij} s_j^1(N) \end{cases} \quad (2)$$

where W_{ij}^0 and Y_{ij} are synaptic weights in the input layer and feedback synapses, respectively. The constant b_i^0 is defined as a bias term, and s^{input} and s^1 are the filtered spiking activities in the input layer and output layer, respectively. The variable s^{input} is calculated based on the following equations as

$$s_j^{input}(t) = \sum_k \kappa(t - t_{jk}^{input}) \quad (3)$$

where t_{jk}^{input} represents the k th spiking time of the input neuron j , and the response kernel is calculated as

$$\kappa(t) = (e^{-t/\tau_L} - e^{-t/\tau_s}) \Theta(t) / (\tau_L - \tau_s) \quad (4)$$

where τ_L and τ_s are long and short time constants, and Θ is the Heaviside step function. The filtered spike trains at apical synapses s^1 is modeled based on the same method. The spiking activities of somatic compartments are based on Poisson processes, whose firing rates are based on a non-linear sigmoid function $\sigma(\cdot)$ for the i th hidden layer neuron as follows:

$$\Phi_i^0(N) = \Phi_{max} \sigma(V_i^0(N)) = \Phi_{max} \frac{1}{1 + e^{-V_i^0(N)}} \quad (5)$$

where Φ_{max} represents the maximum firing rates of neurons.

Plateau Potentials and Weight Updates

Based on the learning algorithm of Guerguiev et al., two phases are alternated to train the network: the forward and target phases as shown in **Figure 3**. In the forward phase $I_i(t) = 0$, while it induces any given neuron i to spike at maximum firing rate or be silent according to the category of the current input image when

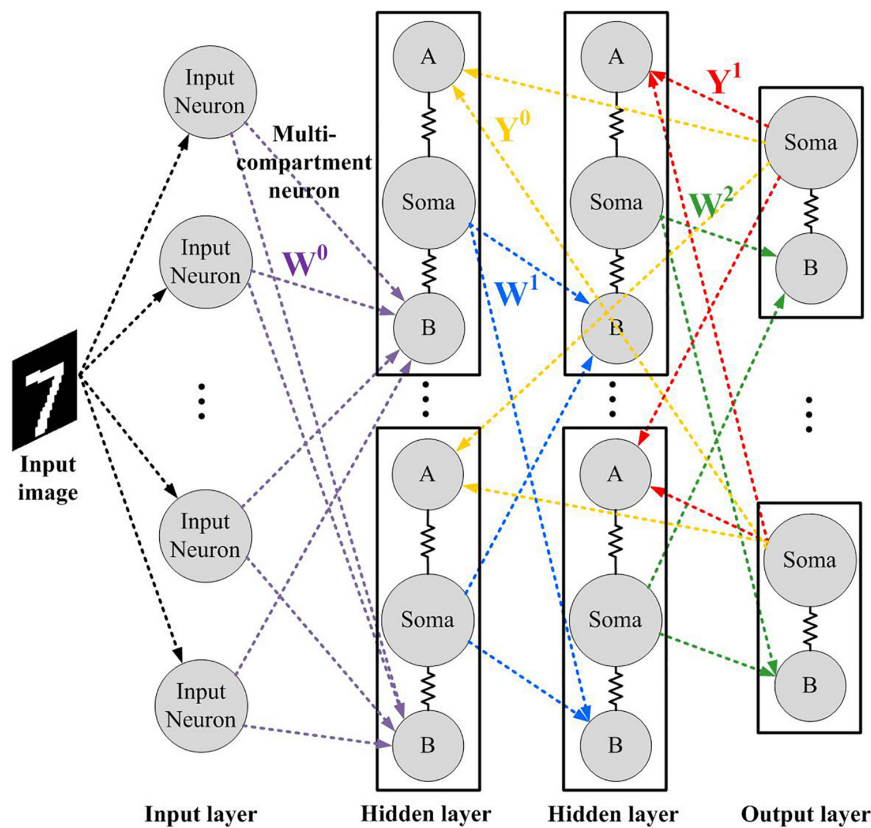


FIGURE 2 | The network architecture in this study. Matrices Y and W represent the feedback and feed-forward synaptic weight matrix, respectively.

the network undergoes target phase. At the end of the forward phase and the target phase, the set of plateau potentials α_t and α_f are calculated, respectively.

At the end of each phase, plateau potentials are calculated for apical dendrites of hidden layer neurons, which are defined as follows

$$\begin{cases} \tau \frac{V_i^{1(N+1)} - V_i^{1(N)}}{\Delta T} = -V_i^{1(N)} + I_i(N) \\ \quad + \frac{g_d}{g_l} (V_i^{1b}(N) - V_i^{1(N)}) \\ V_i^{1b}(N) = \sum_{j=1}^m W_{ij}^{1b0}(N) + b_i^1 \end{cases} \quad (6)$$

where t_1 and t_2 represent the end times of the forward and target phases, respectively. $\Delta t_s = 30$ ms represents the settling time for the membrane potentials, and Δt_1 and Δt_2 are formulated as follows

$$\begin{cases} \Delta t_1 = t_1 - (t_0 + \Delta t_s) \\ \Delta t_2 = t_2 - (t_1 + \Delta t_s) \end{cases} \quad (7)$$

The temporal intervals between plateaus are sampled based on an inverse Gaussian distribution randomly. Although the system computes in phases, the specific length of the phases is not vital, provided there has been a long enough time to integrate the input currents.

Learning With Feedback Driven Plateau Potentials

During the forward phase, an image is presented to the input layer without teaching current at the output layer between time t_0 to t_1 . At t_1 a plateau potential is computed in the hidden layer neurons and the target phase begins. During the target phase the image is also presented into the input layer that also receives teaching current, forcing the correct neuron in the output layer to its maximum firing rate while others are silent. At time t_2 another set of plateau potentials in the hidden layers are computed. Plateau potentials for the end of both the forward and the target phases are calculated as follows

$$\begin{cases} \alpha_i^f = \sigma \left(\frac{1}{\Delta t_1} \int_{t_1 - \Delta t_1}^{t_1} V_i^{0a}(N) dt \right) \\ \alpha_i^t = \sigma \left(\frac{1}{\Delta t_2} \int_{t_2 - \Delta t_2}^{t_2} V_i^{0a}(N) dt \right) \end{cases} \quad (8)$$

where Δt_s represents a time delay of the network dynamics before integrating the plateau, and $\Delta t_i = t_i - (t_{i-1} + \Delta t_s)$. The superscript “ t ” and “ f ” represent target and forward phases, respectively.

The basal dendrites in the hidden layer update the synaptic weights W_0 based on the minimization of the loss function as follows

$$L^0 = \|\phi^{0*} - \phi_{\max} \sigma(\bar{v}^0 f)\|_2^2 \quad (9)$$

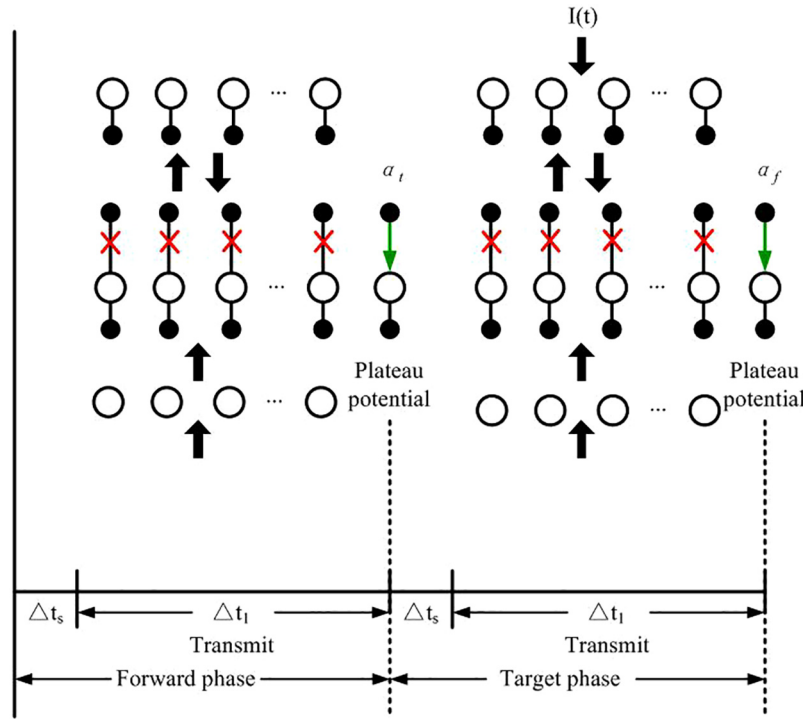


FIGURE 3 | Network computing phases for learning proposed by Guerguiev et al. The green arrows represent the signal transmission from apical dendrite to soma, and red crosses stand for the disconnection between apical dendrite and somatic compartment. The black arrows represent the transmission of spike signals between layers.

And the target firing rate is defined as

$$\phi_i^{0*} = \bar{\Phi}_i^{0f} + \alpha_i^t - \alpha_i^f \quad (10)$$

where the variable and are plateau potentials in the forward and target phases. It should be noted that as long as neural units calculate averages after the network has reached a steady state, and the firing rates of the neurons are in the linear region of the sigmoid function, then we have the following equation for the hidden layer as:

$$\phi_{\max} \sigma(\bar{V}^{0f}) \approx \phi_{\max} \sigma(\bar{V}^{0t}) = \bar{\phi}^{0f} \quad (11)$$

Then the formulation can be obtained as

$$L^0 \approx \|\alpha^t - \alpha^f\|_2^2 \quad (12)$$

And the formulation can be described as follows

$$\begin{cases} \frac{\partial L^0}{\partial W^0} \approx -k_b (\alpha^t - \alpha^f) \phi_{\max} \sigma'(\bar{V}^{0f}) \bar{s}^{inputf} \\ \frac{\partial L^0}{\partial b^0} \approx -k_b (\alpha^t - \alpha^f) \phi_{\max} \sigma'(\bar{V}^{0f}) \end{cases} \quad (13)$$

where the constant k_b is given as

$$k_b = g_b / (g_l + g_b + g_a). \quad (14)$$

In this study, Φ^{0*} is treated as a fixed state for the hidden layer neurons to learn. The synaptic weights of basal dendrites are

updated to descend the approximation of the gradient as follows

$$\begin{cases} W^0 \rightarrow W^0 - \eta^0 P^0 \frac{\partial L^0}{\partial W^0} \\ b^0 \rightarrow b^0 - \eta^0 P^0 \frac{\partial L^0}{\partial b^0} \end{cases} \quad (15)$$

In the target phase the activity is also fixed and no derivatives are used for the membrane potentials and firing rates. The feedback weights are held fixed.

Piecewise Linear Approximation (PWL) for Digital Neuromorphic Computing

Here we simplify the above model for efficient use in neuromorphic architectures. In order to avoid the complicated computation induced by nonlinear functions, the PWL approach is used in this study. Both the functions $\sigma(x)$ and $\sigma'(x)$ are modified based on the PWL method, which can be formulated as follows

$$f_{PWL} = \begin{cases} a_1 x + b_1, & \text{when } x \leq s_1 \\ a_2 x + b_2, & \text{when } s_1 < x \leq s_2 \\ \dots \\ a_i x + b_i, & \text{when } x > s_{i-1} \end{cases} \quad (16)$$

where a_i and b_i are the slope and intercept of the modified PWL function f_{owl} , respectively ($i = 1, 2, \dots, n$). Since the range of the segment points are constrained, an exhaustive search algorithm is used in the determination of the PWL functions.

TABLE 1 | Parameter values of the PWL methods.

$\sigma(x)$	A	b	Condition
$i = 1$	0.0078125	0.05	$x \leq -3.4$
$i = 2$	0.0625	0.24	$-3.4 < x \leq -1.3$
$i = 3$	0.25	0.5	$-1.3 < x \leq 1.3$
$i = 4$	0.0625	0.76	$1.3 < x \leq 3.4$
$i = 5$	0.0078125	0.95	$3.4 < x$
$i = 6$	0	0.9999	$\sigma(x) \leq 0$
$i = 7$	0	0.0001	$\sigma(x) \geq 1$
$\sigma'(x)$	A	b	Condition
$i = 1$	0.0078125	0.05	$x \leq -3.2$
$i = 2$	0.03125	0.15	$-3.2 < x \leq -2$
$i = 3$	0.0625	0.25	$-2 < x \leq 0$
$i = 4$	-0.0625	0.25	$0 < x \leq 2$
$i = 5$	-0.03125	0.15	$2 < x \leq 3.2$
$i = 6$	-0.0078125	0.05	$x > 3.2$
$i = 7$	0	0.0001	$\sigma'(x) \leq 0$

The determination of the coefficient values are based on an error evaluation criterion as follows

$$CF_{RE} = \frac{1}{n} \sqrt{\sum_{i=1}^n \frac{(f_{ori}(i) - f_{PWL}(i))^2}{f_{ori}(i)^2}} \quad (17)$$

where n represents the total sampling points, and f_{ori} represents the original function. If the modified function cannot meet the accuracy requirement represented by CF_{RE} , its segment number will be added by 1 until it can be guaranteed. Since the multiplication operation is replaced by "adder" and "shifter" in the proposed study, the coefficient value a_i in the PWL functions should be a power of 2 (for example: 1, 2, 4 or 0.5, 0.25, etc.). The parameter values of the PWL methods are listed in **Table 1**. The PWL functions are depicted in **Figure 4**.

Binarization for Filtered Spike Trains

The digital neuromorphic algorithm requires less multiplication operations. Therefore, in this study we use the Otsu's

thresholding method to binarize the filtered spike trains, which can iterate all possible threshold values and compute the expansion measure of each pixel level of the threshold (Otsu, 1978). Therefore, each pixel will fall in either foreground or background. Firstly, separate all the pixels into two clusters based on the threshold as follows

$$\begin{cases} q_1(t) = \sum_{i=1}^t p(i) \\ q_2(t) = \sum_{i=t+1}^L p(i) \end{cases} \quad (18)$$

where p represents the image histogram. Secondly, the mean of each cluster is calculated by the formulation as follows

$$\begin{cases} \mu_1(t) = \sum_{j=1}^t \frac{j \cdot p(j)}{q_1(t)} \\ \mu_2(t) = \sum_{j=t+1}^L \frac{j \cdot p(j)}{q_2(t)} \end{cases} \quad (19)$$

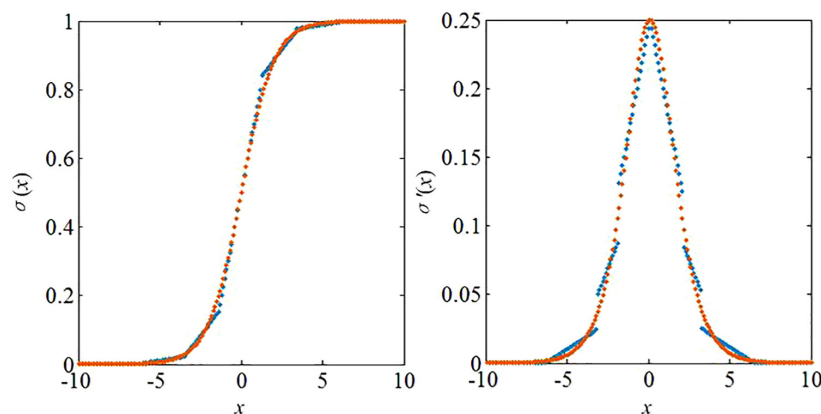
Thirdly, calculate the individual class variance as follows

$$\begin{cases} \lambda_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{p(i)}{q_1(t)} \\ \lambda_2^2(t) = \sum_{i=t+1}^L [i - \mu_2(t)]^2 \frac{p(i)}{q_2(t)} \end{cases} \quad (20)$$

Fourthly, square the difference between the means formulated as follows

$$\begin{aligned} \lambda_b^2(t) &= \lambda^2 - \lambda_w^2(t) \\ &= q_1(t) [1 - q_1(t)] [\mu_1(t) - \mu_2(t)]^2 \end{aligned} \quad (21)$$

where λ_b , λ , and λ_w represent between-class variance, total class variance and within-class variance, respectively. Finally, the formulation can be maximized and the solution is t that is maximizing $\lambda_b^2(t)$.

**FIGURE 4** | PWL functions in the proposed algorithms.

Considerations for Training With Low Bitwidth Weights

Neuromorphic hardware is largely made out of arithmetic elements and memories. Multipliers are the most space and power hungry arithmetic elements of the digital neuromorphic implementation. The realization of a deep neural network is mainly dependent on matrix multiplications. The key arithmetic operation is the multiply-accumulate operation. The reduction of the precision of the multipliers, especially for the weight matrix, is vital for the efficient realization of deep neural networks. Recent researches have focused on the reduction of model size and computational complexity by using low bitwidth weights of neural networks (Courbariaux et al., 2014). Other neuromorphic hardware systems implement bistable synapses based on a 1-bit weight resolution, which is shown to be sufficient for memory formation (Bill, 2010). However, the models do not only use spike timings, but also use additional hardware resources to read the postsynaptic membrane potential (Sjöström et al., 2001). Therefore, this study trains the proposed DEP algorithm using dynamic fixed point representation. In dynamic fixed point, each number is represented as follows

$$(-1)^s \cdot 2^{-FL} \sum_{i=0}^{B-2} 2^i \cdot x_i \quad (22)$$

where B represents the bit-width, s the sign bit, FL is the fractional length, and x the mantissa bits.

The proposed algorithm is presented in **Figure 5**. In the pseudo code, the synaptic weight matrix W is the input of the algorithm. $Total_bit$ represents the total bit width of the fixed-point number, and IF_bit is the integer bitwidth. The fractional bitwidth is represented by LF_bit . The integer and fractional parts are represented by W_IF and W_LF . The binary integer and fractional parts are represented by W_IF_bit and W_LF_bit , respectively. The symbol bit is represented by W_s , and R_max defines the fault-tolerant ratio. The error rate refers to the difference between the binary number and the original decimal number divided by the original decimal number. If the error rate exceeds the defined fault-tolerant rate, a specialized process will be used for the binary number. Since the large error occurs in the situation when the considered number is close to 0, this number will be set to 0 if the error rate exceeds R_max . The term W is an $a \times b$ synaptic weight matrix to be trained. The first loop is in the line 2. This loop is in the line 2, which is the row loop of the matrix. The second loop is in the line 3, which is the column loop of the matrix. There are two judgments in the proposed algorithm. The first judgment is to determine the symbol bit. If it is negative, then the symbol is 1. If it is positive, then the symbol is 0. The second judgment is to determine the positive and negative when the binary number is converted to decimal number. If the sign bit is 1, it is negative. And it is positive when the sign bit is 0. The third judgment is to consider the error rate between the newly converted number and the original number. If the error rate exceeds the fault-tolerant ratio, the newly converted number will be replaced by 0 for efficient calculation on neuromorphic systems. Finally the updated synaptic weight matrix W_new is

Algorithm: Training with dynamic fixed point representation

Input: W , **Output:** W_new ;

W is $a \times b$ matrix.

For $i=1:a$

For $j=1:b$

If $W(i,j) < 0$

$W_s = 1$;

Else

$W_s = 0$;

End

$W_IF_bit = W_IF \rightarrow bit$;

$IF_bit = length(W_IF_bit)$;

$LF_bit = Total_bit - IF_bit - 1$;

$W_LF_bit = (W_LF, LF_bit) \rightarrow bit$;

$W_new(i,j) = W_IF_bit \rightarrow dec + W_LF_bit \rightarrow dec$;

If $W_s = 1$

$W_new(i,j) = -W_new(i,j)$;

End

If $(W_new(i,j) - W(i,j)) / W(i,j) > R_max$

$W_new(i,j) = 0$;

end

End

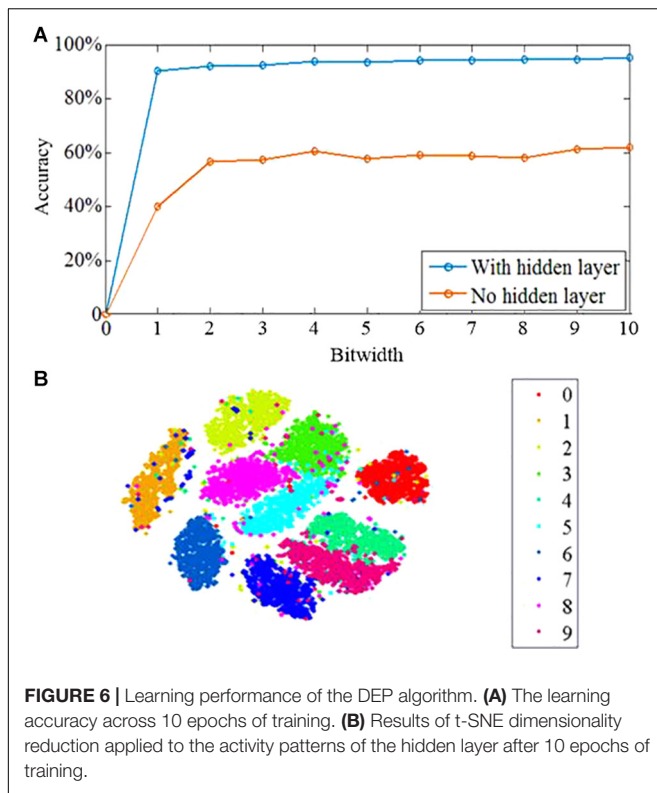
End

FIGURE 5 | Pseudocode of the algorithm for training with dynamic fixed point representation.

output by the processing of the proposed algorithm. By using the proposed algorithm, the memory usage on hardware can be optimized and the energy efficiency of neuromorphic systems can be further improved.

RESULTS

To demonstrate the effectiveness of the proposed learning algorithm, the standard Modified National Institute of Standards and Technology database (MNIST) benchmark is employed. The MNIST dataset contains 70,000 28×28 images of handwritten digits. The image number in the training and testing sets are 60,000 and 10,000, respectively. The dataset is divided into 10 categories for 10 integers 0–9, and each image has an associated label. We trained the networks with no hidden layer, with one hidden layer and two hidden layers on the 60,000 MNIST training images for 10 epochs, and tested the classification accuracy using the 10,000 image test set. As shown in **Figure 6A**, the network with no hidden layer has poor classification performance of 62.1%. In contrast, the three-layer network with hidden layer has an accuracy of 95.1% by the 10th epoch. The proposed network can take advantage of the multi-layer architecture to enhance the learning performance, which is the critical characteristics of deep learning (Bengio and LeCun, 2007). Another critical characteristic of deep learning is the capability to generate representations, which obtains task-related information and ignores irrelevant



sensory details (LeCun et al., 2015; Mnih et al., 2015). The t-distributed stochastic neighbor embedding algorithm (t-SNE) is used to investigate the information abstraction of the proposed algorithm. The t-SNE algorithm can reduce the dimensionality of data with the preservation of local structure and nonlinear manifolds in high-dimensional space. It is a powerful approach to visualize the structure of high-dimensional data (Maaten and Hinton, 2008). The t-SNE algorithm is applied to the hidden layer, which shows that the categories are better segregated with only a small amount of splitting or merging of category clusters as shown in **Figure 6B**. Therefore, the proposed algorithm has the capability of learning the developing representations in the hidden layer, in which the categories are quite distinct. It reveals that the proposed algorithm can be applied in a deep learning framework. In addition, the proposed algorithm relies on the phenomenon of feedback alignment, in which the feed-forward system comes to align with the feedback weights so that a useful error signal is provided.

The proposed DEP learning algorithm in a network with one hidden layer trained on permutation invariant MNIST is explored, although it can be generalized to other datasets in theory. Rather than seeking for the optimized classification performance, the equivalent non-spiking neural networks trained by standard BP and random BP are compared with the proposed algorithm, with the parameters tuned to obtain the highest classification accuracy in the current classification task. Weight updates are conducted during each digit input into the spiking neural network, which is different from the batch gradient descent that performs weight updates once per the entire dataset. As

shown in **Figure 7**, the DEP algorithm requires fewer iterations of the dataset to obtain the peak classification performance in comparison with the two alternative methods. The reason is that the spiking neural network with DEP algorithm can be updated multiple times during each input, which results in faster convergence of learning. In addition, for the equivalent computational resources, online learning with gradient descent strategy has the capability to deal with more data samples and requires less on-chip memory for implementation (Bottou and Cun, 2004). Therefore, for the same number of calculation operations per unit time, online gradient-descent-based learning converges faster than batch learning. Since potential applications of neuromorphic hardware is with real-time streaming data, it is essential for the online learning with DEP algorithm.

In order to further demonstrate the learning performance of the proposed DEP algorithm, a comparison between the proposed DEP algorithm with two layers and the SNN with point LIF neuron model is presented. As shown in **Figure 8A**, the learning accuracy of the SNN model with dendrites, i.e., the proposed DEP algorithm, is higher than the conventional SNN with point neuron model. Besides, we further apply the DEP algorithm in the feature detection tasks to see whether the proposed algorithm could also learn feature detection maps from continuous sensory streams. Previous study has shown that SNN models can detect features from background activities (Masquelier et al., 2008). In order to provide a good benchmark for the proposed DEP algorithm on the feature detection task, the ability of the DEP algorithm is examined for feature detection tasks. In this task, there are eight categories, and each category represents on direction, including 0° , 22.5° , 45° , 67.5° , 90° , 112.5° , 135° , and 157.5° . Each image consists of 729 (27×27) pixels. Besides, 10% pixels are randomly

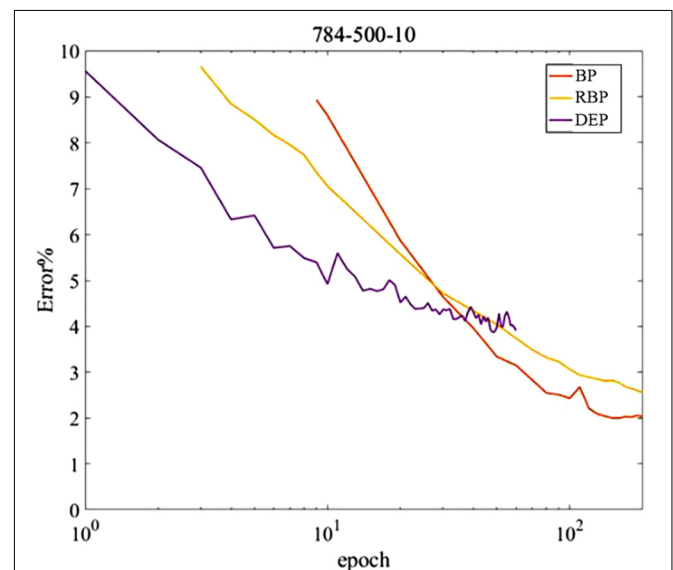


FIGURE 7 | MNIST classification error based on spiking neural network with DEP learning rule and fully connected artificial neural networks with backpropagation (BP) and random backpropagation (RBP) learning rules.

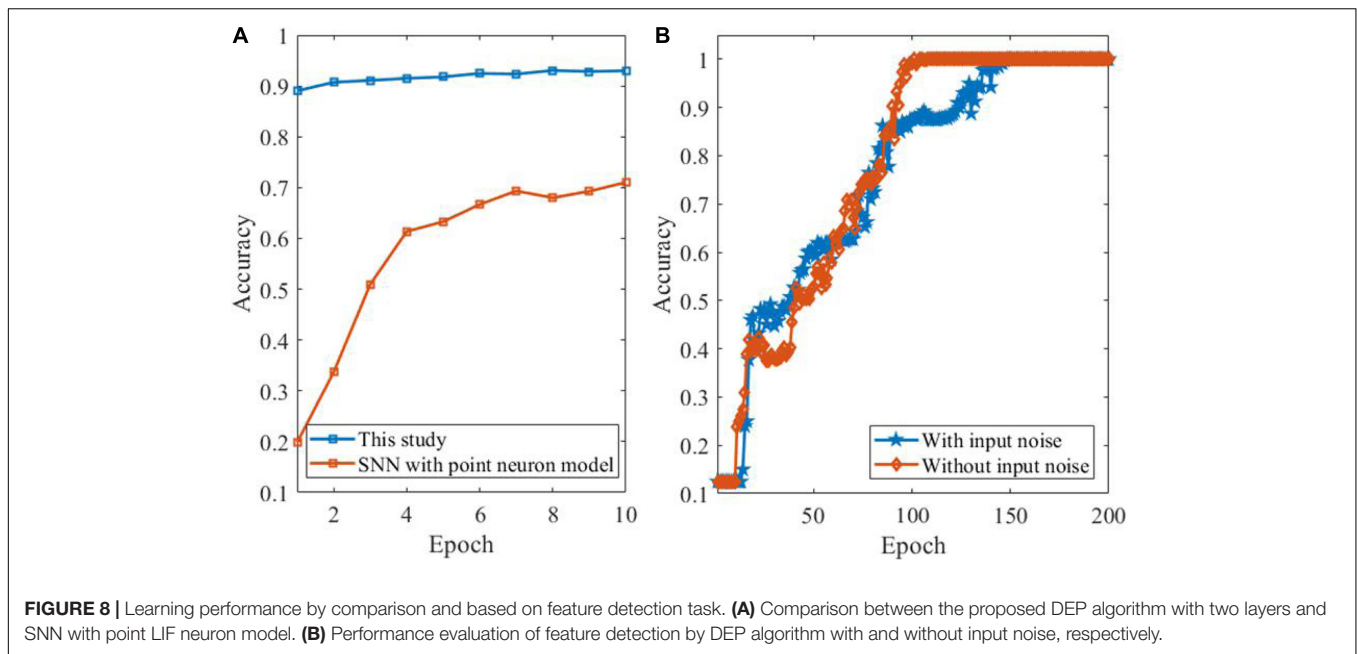


FIGURE 8 | Learning performance by comparison and based on feature detection task. **(A)** Comparison between the proposed DEP algorithm with two layers and SNN with point LIF neuron model. **(B)** Performance evaluation of feature detection by DEP algorithm with and without input noise, respectively.

selected to add Gaussian noise to make the data set with input noise. **Figure 8B** shows the learning performance of the DEP algorithm. It reveals that the DEP algorithm can successfully detect feature patterns contaminated by background noise using spike-based framework.

As shown in **Figure 9**, learning on neuromorphic system can be energy efficient by using the proposed DEP algorithm, because only active connections in the network induce synaptic operations (SynOps) operation. In order to show the learning efficiency, the number of multiply-accumulate (MAC) operations using the BP algorithm is compared with SynOps number with the proposed algorithm. This advantage is critical and promising for neuromorphic computing because SynOps in a

dedicated neuromorphic system use much less power than MAC operations on a GPU platform. The learning accuracy of the proposed algorithm increases quickly but the final accuracy is lower than an ANN.

As shown in **Figure 10**, the response of the proposed DEP algorithm after stimulus onset is one synaptic time constant. It leads to 11% error and improves as the spikes number of the neurons in the output layer increases. Classification using the first spike induced less than 20 k SynOps events, most of which exist between the input and hidden layer. In the state-of-the-art neuromorphic system, the energy consumption of a synaptic operation is around 20 pJ (Merolla et al., 2014; Qiao et al., 2015). On such neuromorphic system, single spike classification based on the proposed network can potentially induce 400 pJ, which is superior to the state-of-the-art work in digital neuromorphic hardware ($\approx 2 \mu\text{J}$) at this accuracy (Esser et al., 2016) and potentially 50,000 more efficient than current GPU technology. In addition, an estimation of the power consumption during training by using BP and the proposed DEP algorithm is also presented according to **Figure 9**. It reveals that about 10^{11} SynOps is cost by the end of 60 epochs, therefore about 33 mJ is cost in an epoch during training. Previous study has demonstrated that 1,000–5,000 mJ will be cost by BP algorithm on conventional GPU platform (Rodrigues et al., 2018). Therefore, there is a 96.7–99.3% reduction for the power consumption by the proposed DEP algorithm during training. The reasons for the low energy cost can be divided into three aspects. Firstly, the segregated dendrite can generate a plateau potential within 50–60 ms, which determines the training time of the proposed network. The training time can be thereby reduced in this way, which can cut down the number of spikes with the decreasing of the training time for each image. Secondly, the conventional BP algorithm induces a trend to make neurons spike with maximum firing rate, and induces synchronization

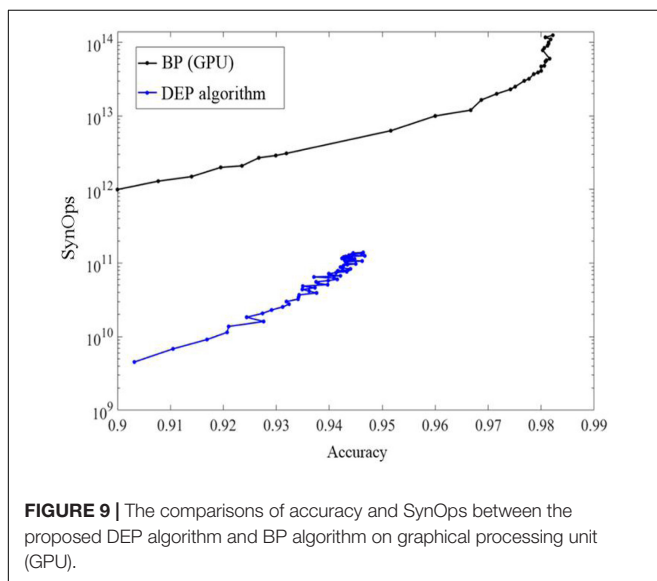
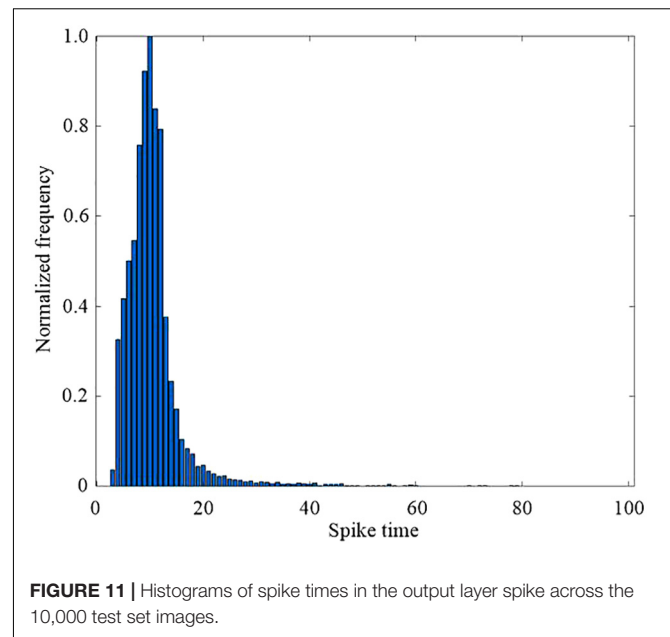
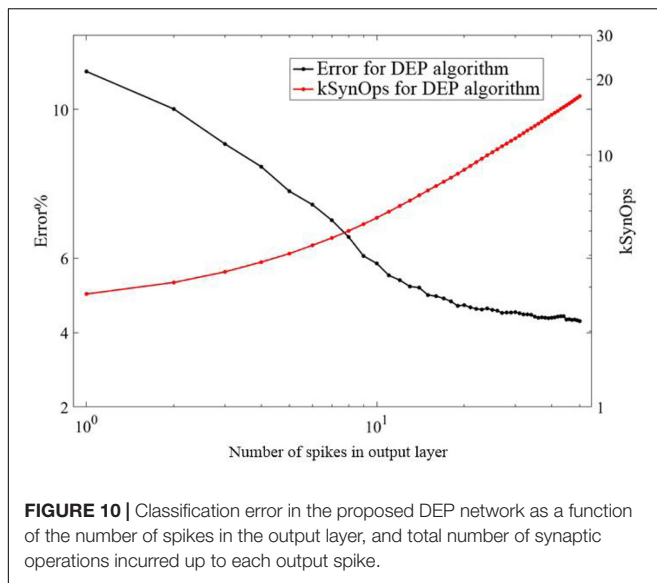


FIGURE 9 | The comparisons of accuracy and SynOps between the proposed DEP algorithm and BP algorithm on graphical processing unit (GPU).



within layers. This means a larger number of spikes. Thirdly, the communication between layers in the proposed algorithm uses a Poisson filter, and Φ_{max} is set to be 0.2. These results suggest that the proposed DEP learning algorithm can take full use of the spiking dynamics, with the learning accuracy comparable to the spiking network that is trained specifically for single spike recognition in previous study (Mostafa, 2017).

Figure 11 shows the distribution of spike times in the output layer, which is the times at which the proposed SNN makes a decision for all the 10,000 test set images. The proposed SNN with DEP algorithm makes a decision after most of the hidden layer neurons have spiked. The network is thus able to make more accurate and robust decisions about the input images, based on the plateau potentials generated by the dendrites in the proposed DEP algorithm.

As shown in **Figure 12**, 30 neurons in the hidden layer are selected randomly to explore the selectivity for 10 categories of MNIST data set. The negative log probability for each of the 30 neurons to spike for each of the 10 categories is explored, which means the negative log probability for a neuron to participate in the classification of a specified category. Probability is calculated from the response of the SNN to the 10,000 test digits. It reveals that some neurons are highly selective, while most of the neurons are more broadly tuned. Some of the neurons are mostly silent, but all the neurons in the SNN model contribute to at least one category of classification with the 10,000 test digits. In other word, neurons are typically broadly tuned and contribute to the classification of more than one categories.

We further investigate the necessary bit widths of the fixed-point and dynamic fixed-point, respectively. The bit width of the integer part using the fixed-point calculation is set to 8 to avoid the overflow problem during computation. In contrast, the dynamic fixed-point is not required to determine the bit width of either integer or fractional part. As shown in **Figure 13**, the fixed-point representation of the fractional part requires 14 bits to obtain a satisfied learning performance

that exceeds 90%. Therefore, the satisfied total bit width for fixed-point representation is 22 bit (8 bit for integer part and 14 bit for fractional part). The dynamic fixed-point representation just needs 16 bits to realize high-performance learning. Therefore, the dynamic fixed-point representation in the proposed algorithm provides an efficient approach to reduce the computational hardware resource cost and power consumption for neuromorphic computing.

Figure 14 shows the digital neuromorphic architecture at the top level, which contains an input layer, a hidden layer with five physical neural processors, and an output layer with 10 physical neural processors. The input layer and hidden layer are all implemented to use time-multiplexing. The global counter processors the time-multiplexed input neurons and hidden layer neurons sequentially. The FSM module represents the finite-state machine which controls the timing procedure of the whole neuromorphic system. Three parts are contained in the neuron processor in the hidden layer, which are apical dendrite unit, soma unit and basal dendrite unit. The neuron processor in the output layer consists of two parts that are apical dendrite unit and soma unit. The input of the teaching current $I(t)$ is also mastered by the FSM. The green arrows represent the synaptic connections with learning mechanisms, and black arrows describe the invariant synaptic coupling.

The detailed description of the FSM is shown in **Figure 15A**. There are eight states in the FSM diagram, including idle, first time delay, forward phase, first plateau potential (PP) computation, second time delay, target phase, second PP computation and weight updating. By using the FSM controller, the digital neuromorphic system can operate in high performance with definite timing sequence. **Figure 15B** depicts the internal architecture of the time-multiplexed system. It consists of a physical input neuron, two physical hidden neurons, a global

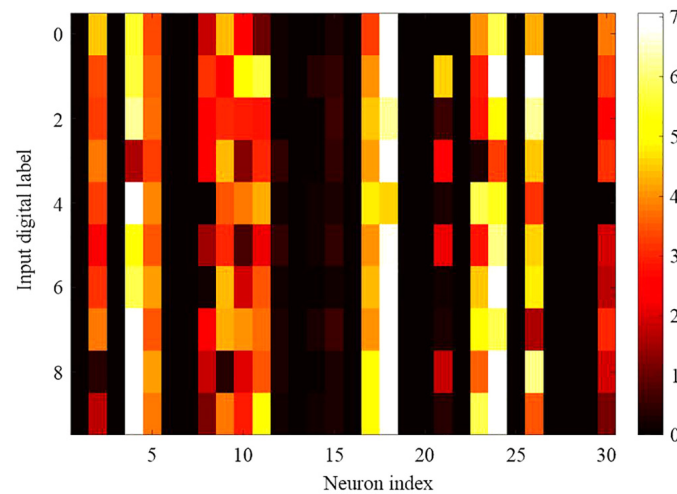


FIGURE 12 | Selectivity and tuning properties of 30 randomly selected hidden neurons in the proposed SNN network with DEP algorithm. It is plotted by heat map with color called YlOrRd, whose color gradually changes across yellow, orange, and red.

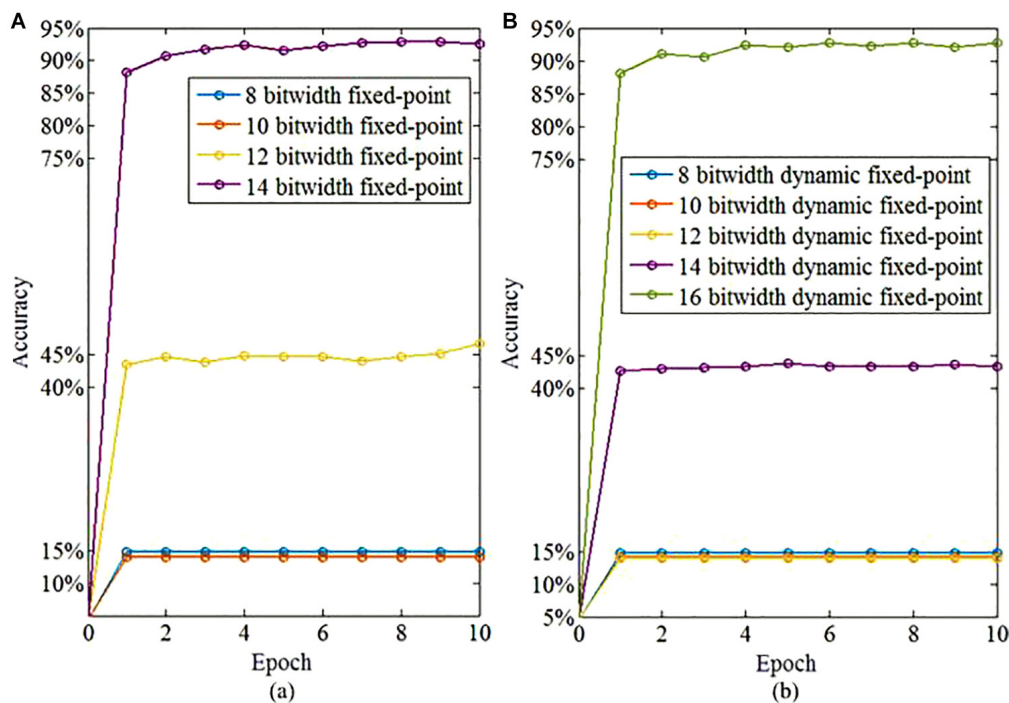


FIGURE 13 | The learning accuracy based on fixed-point and dynamic fixed-point representations. **(A)** The learning accuracy based on fixed-point representation of fractional part with different bitwidth. **(B)** The learning accuracy based on dynamic fixed-point representation with different bitwidth.

counter, and two weight buffers for each physical hidden neuron. The global counter processes the time-multiplexed physical input and hidden neurons sequentially. The weight buffers store the synaptic weights of the physical neurons. The input digit signals remains available until all the time-multiplexed physical neurons finish their computation. We can also employ the pipeline architecture, by which the maximum operating frequency of the neuromorphic system can be further enhanced.

DISCUSSION

This study presents a multi-layer feed-forward network architecture using segregated dendrites and the corresponding two-phase learning scheme. Specifically, a piecewise linear approximation and a dynamic fixed-point representation are first introduced in the dendritic learning framework for cost and energy efficient neuromorphic computing. It relies on the

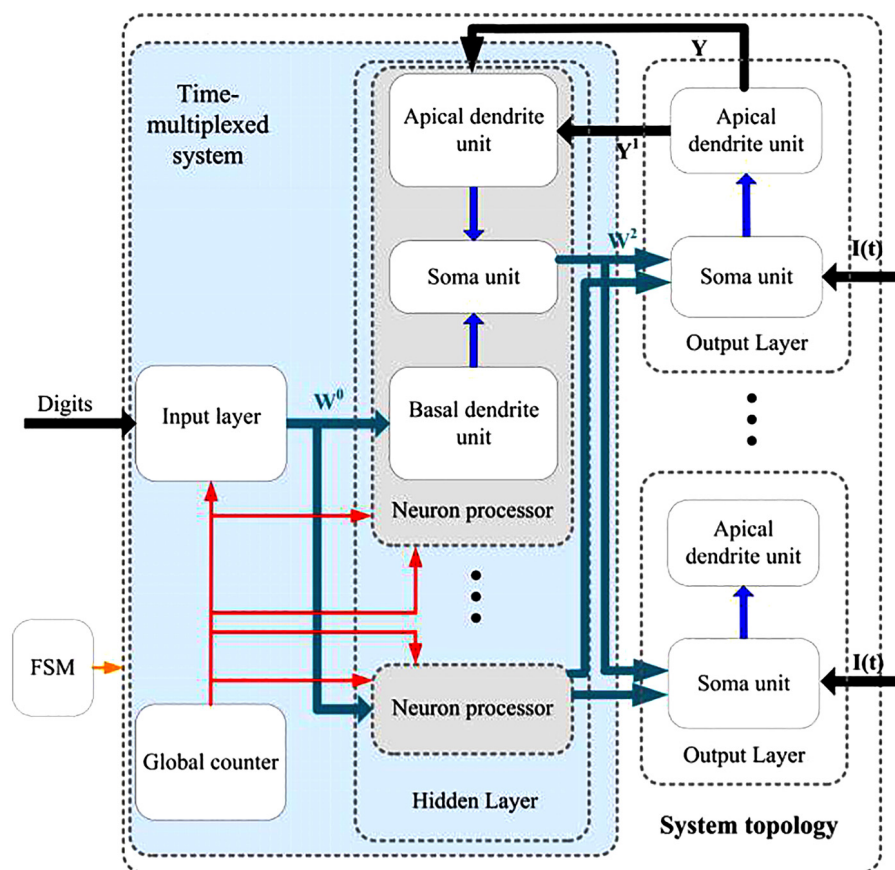


FIGURE 14 | Top-level entity for the neuromorphic architecture of the proposed learning algorithm.

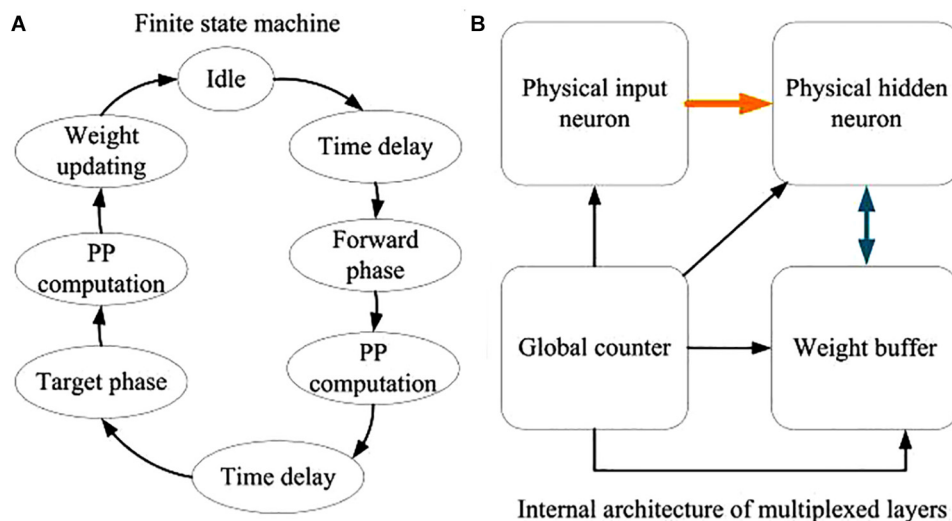


FIGURE 15 | Descriptions of the digital neuromorphic architecture. **(A)** The FSM diagram. **(B)** The internal architecture of the time-multiplexed system.

feedback alignment phenomenon, in which the feed-forward weights are aligned with the feedback weights to provide useful error signals for learning. The model is optimized for the efficient neuromorphic realization by using the PWL approximation,

as well as the binarization for synaptic events. A dynamic fixed-point representation technique is further presented to optimize the proposed DEP algorithm. It reveals that the proposed algorithm with hidden layers can induce higher

learning performance, which means that it contains the deep learning capability. In addition, the energy efficient property is proven by comparison with the conventional artificial neural network with BP algorithm. The reasons for this result are likely due to two reasons. First, the gradient descent algorithm suffers from the local optimization problem. When the local optimization is realized, the global optimization cannot be obtained. Spikes emanating from error-coding neurons will be so sparse toward the end of the training that it will prevent the successful adjustments of the weight. The low learning rate will aggravate this problem. A scheduled adjustment of the error neuron sensitivity may solve this problem. Second, the proposed algorithm has not fully utilized the nonlinear dynamics of the neural dendrite. The dynamics of the dendritic compartment have the capability of predicting the dynamics, which may help to improve the performance when considering the dendritic prediction feature. These two methods of modifications, as well as more complicated learning rules, such as momentum or learning rate decay, are left for future work. The output layer neurons spike after the spiking activities of the most neurons in the hidden layer, thus induce a more accurate and robust classification results. The broadly tuned property of the neurons in the hidden layer of the proposed SNN shows that the proposed DEP algorithm can engage each neuron to participate in the classification task. In addition, it shows the superior performance by using the proposed dynamic fixed-point representation by comparing it with the traditional fixed-point computation, which shows that the proposed method can reduce the hardware resource cost considerably. Therefore, our study demonstrates a biologically plausible learning algorithm in a neuromorphic architecture, and realizes the efficient learning by using the DEP approach. In summarize, the proposed DEP algorithm has four aspects of advantages. Firstly, the proposed DEP algorithm cost less SynOps number in comparison with the conventional BP algorithm as shown in **Figure 8**. It means less power consumption can be realized on neuromorphic hardware. Secondly, faster learning speed can be achieved by the DEP algorithm shown in **Figure 7**, which is meaningful for on-chip online learning. Thirdly, the solution of credit assignment by dendrites is a vital mechanism for learning in human brain. Therefore, the proposed DEP algorithm is more biologically plausible, which is also a significant ambition of neuromorphic computing. Fourthly, the proposed DEP algorithm is more useful for the online learning with network architecture using more than one layer. As shown in Figure, single point neuron model is not suitable for learning with gradient descent when the network layer number increasing to two.

In the field of neuromorphic computing, neuromorphic systems with on-line learning ability provide a platform to develop brain-inspired learning algorithms, which strive to emulate in digital or analog technologies human brain properties. Online learning requires to be realized based on the input of asynchronous and event-based sequential data flow. Since neuromorphic computing supports continual and lifelong learning naturally, this study presents a SNN model that can deal with the asynchronous event-based spatio-temporal information, which is applicable for neuromorphic systems directly. It provides a novel view for neuromorphic online learning and continual

learning, which is meaningful to bridge the gap between neuroscience and machine intelligence. Previous studies have presented a number of neuromorphic systems equipped with synaptic plasticity for general-purpose sensorimotor processors and reinforcement learning (Nefci, 2013; Qiao, 2015; Davies et al., 2018). However, current neuromorphic computing ignores the learning capability to further improve the deep learning performance. Inspired by other neuromorphic studies, more low-power and high-speed techniques can be considered in the future work to obtain a better learning effect.

Previous studies have proposed new algorithms, including attention-gated reinforcement learning (AGREL) and attention-gated memory tagging (AuGMEnT) learning rules, explaining the mechanism of the reinforcement learning optimization in a biologically realistic manner using synapses in deep networks (Roelfsema and Ooyen, 2005; Rombouts et al., 2015). The feedback coupling strength is proportional to the feed-forward strength in these models, which means the learning principles are computationally equivalent to the error back-propagation. It indicates the human brain can solve the credit-assignment problem in a manner that is equivalent to deep learning. However, AGREL algorithm uses the top-down probabilistic model to compute rather than the description and representation of learning from the neural dynamics point of view. There is also no bottom-to-top modeling using spiking neurons in AuGMEnT algorithm. Thus, these two algorithms cannot be employed in neuromorphic computing. Interestingly, we can combine these two algorithms with the presented DEP algorithm to improve the learning performance further.

Efficient learning to solve the credit assignment problem is helpful for the performance improvement of deep learning. This study presents the DEP algorithm for neuromorphic learning, which is meaningful for the communities of both neuromorphic engineering and deep learning. Recently, neuromorphic computing has wide applications. Neuromorphic vision sensors capture the features of biological retina, which has changed the landscape of computer vision in both industry and academia (Chen et al., 2019; Zhou et al., 2019). Although neuromorphic systems with deep learning capability are still in research phases, the development of neuromorphic computing is calling for more biologically realistic processing strategies. Looking forward, with such systems with learning ability, the bridges between machine and biological learning can translate into adaptive and powerful embedded computing systems for a wide category of applications, such as object recognition, neuro-robotic control, and machine learning.

CONCLUSION

This paper presented a biologically meaningful DEP algorithm with dynamic fixed-point representation, as well as its digital neuromorphic architecture on LaCSNN. The PWL approximation method and the binarization approach for synaptic events are used in the proposed algorithm for the optimization of efficient implementation. Experimental results show that the learning performance of the proposed DEP algorithm can be improved by adding a hidden layer, which

shows the deep learning capability of DEP. Different levels of dendrite segregation will influence the learning accuracy of the network, and the manners of the synaptic feedback connections also play vital roles in the learning performance. By using the fixed-point representation in this work, the hardware resource cost can be cut down by reducing the bit width of the computational elements. This study provides a bridge between the biological learning and neuromorphic learning, which can be used in the applications including object recognition, neuro-robotic control, and machine learning.

DATA AVAILABILITY STATEMENT

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

REFERENCES

- Bengio, Y., and LeCun, Y. (2007). Scaling learning algorithms towards AI. *Large Scale Kernel Mach.* 34, 1–41.
- Bengio, Y., Lee, D. H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv [Preprint]*. arXiv:1502.04156
- Bill, J. (2010). Compensating inhomogeneities of neuromorphic VLSI devices via short-term synaptic plasticity. *Front. Comput. Neurosci.* 4:129. doi: 10.3389/fncom.2010.00129
- Bittner, K. C. (2015). Conjunctive input processing drives feature selectivity in hippocampal CA1 neurons. *Nat. Neurosci.* 18:1133. doi: 10.1038/nn.4062
- Bono, J., and Clopath, C. (2017). Modeling somatic and dendritic spike mediated information fusion for pedestrian detection with neuromorphic vision sensors. *Front. Neurobot.* 13:10. doi: 10.3389/fnbot.2019.00010
- Courbariaux, M., Bengio, Y., and David, J. P. (2014). Training deep neural networks with low precision multiplications. *arXiv [Preprint]*. arXiv:1412.7024
- Davies, M., Srinvasa, N., and Lin, T. H. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/mm.2018.112130359
- Esser, S. K., Appuswamy, R., and Merolla, P. (2015). Backpropagation for energy-efficient neuromorphic computing. *Adv. Neural Inf. Process. Systems* 28, 1117–1125.
- Esser, S. K., Merolla, P. A., Arthur, J. V., and Cassidy, A. S. (2016). Convolutional networks for fast, energy efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446.
- Guerguiev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites. *eLife* 6:e22901.
- Indiveri, G., Corradi, F., and Qiao, N. (2015). “Neuromorphic architectures for spiking deep neural networks,” in *Proceedings of the 2015 IEEE International Electron Devices Meeting (IEDM)*, Washington, DC, 4–2.
- Katharina, A., Henning, S., and Susanne, S. (2016). Inhibition as a binary switch for excitatory plasticity in pyramidal neurons. *PLoS Comput. Biol.* 12:e1004768. doi: 10.1371/journal.pcbi.1004768
- Lansdell, B. J., and Kording, K. P. (2019). Spiking allows neurons to estimate their causal effect. *bioRxiv [Preprint]*. doi: 10.1101/253351
- Lansdell, B. J., Prakash, P. R., and Kording, K. P. (2019). Learning to solve the credit assignment problem. *arXiv [Preprint]*. arXiv:1906.00889
- Larkum, M. (2013). A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends Neurosci.* 36, 141–151. doi: 10.1016/j.tins.2012.11.006

AUTHOR CONTRIBUTIONS

SY, TG, and BL-B developed the theoretical approach for DEP algorithm with spiking neurons. TG implemented the source code. JW and BL revised the manuscript and made critical suggestions on this work. SY wrote the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This study was funded partly by the National Natural Science Foundation of China with grant numbers (Grant Nos. 62071324 and 62006170) and partly by China Postdoctoral Science Foundation (Grant No. 2020M680885).

- Larkum, M. E., Zhu, J. J., and Sakmann, B. (1999). A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature* 398, 338–341. doi: 10.1038/18686
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444.
- Lee, D. H., Zhang, S., Fischer, A., and Bengio, Y. (2015). “Difference target propagation,” in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (New York, NY: Springer International Publishing), 498–515.
- 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
- Liao, Q., Leibo, J. Z., and Poggio, T. (2016). How important is weight symmetry in backpropagation. *arXiv [Preprint]*. arXiv: 1510.05067
- 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, 1–10. doi: 10.1016/j.artint.2018.03.003
- Maaten, L., and Hinton, G. (2008). Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, 2579–2605.
- Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2008). Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains. *PLoS One* 3:e1377. doi: 10.1371/journal.pone.0001377
- Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., and Cassidy, A. S. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642
- Mnih, V., Kavukcuoglu, K., and Silver, D. (2015). Human-level control through deep reinforcement learning. *Nature* 518:529.
- Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235.
- Neftci, E. (2013). Synthesizing cognition in neuromorphic electronic systems. *Proc. Natl. Acad. Sci. U.S.A.* 110, 3468–3476.
- Neftci, E. O., Augustine, C., and Paul, S. (2017). Event-driven random backpropagation: enabling neuromorphic deep learning machines. *Front. Neurosci.* 11:324. doi: 10.3389/fnins.2017.00324
- Otsu, N. (1978). A threshold selection method from gray-scale histogram. *IEEE Trans. Syst. Man Cybern.* 8, 62–66. doi: 10.1109/tsmc.1979.4310076
- Qiao, N. (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
- Qiao, N., Ning, H., and Corradi, F. (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
- Richards, B. A., and Lillicrap, T. P. (2019). Dendritic solutions to the credit assignment problem. *Curr. Opin. Neurobiol.* 54, 28–36. doi: 10.1016/j.conb.2018.08.003
- Rodrigues, C. F., Riley, G., and Luján, M. (2018). “SyNERGY: an energy measurement and prediction framework for convolutional neural networks

- on Jetson TX1[C],” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)*, 2018, Turin, 375–382.
- Roelfsema, P. R., and Ooyen, A. (2005). Attention-gated reinforcement learning of internal representations for classification. *Neural Comput.* 17, 2176–2214. doi: 10.1162/0899766054615699
- Rombouts, J. O., Bohte, S. M., and Roelfsema, P. R. (2015). How attention can create synaptic tags for the learning of working memories in sequential tasks. *PLoS Comput. Biol.* 11:e1004060. doi: 10.1371/journal.pcbi.1004060
- 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
- Schmolesky, M. T., Weber, J. T., Zeeuw, C. I. D., and Hansel, C. (2002). The making of a complex spike: ionic composition and plasticity. *Ann. N. Y. Acad. Sci.* 978, 359–390. doi: 10.1111/j.1749-6632.2002.tb07581.x
- Sjöström, P. J., Turrigiano, G. G., and Nelson, S. B. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron* 32, 1149–1164. doi: 10.1016/s0896-6273(01)00542-6
- Spratling, M. W. (2002). Cortical region interactions and the functional role of apical dendrites. *Behav. Cogn. Neurosci. Rev.* 1, 219–228. doi: 10.1177/1534582302001003003
- 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
- Wilmes, K. A., Sprekeler, H., and Schreiber, S. (2016). Inhibition as a binary switch for excitatory plasticity in pyramidal neurons. *PLoS Comput. Biol.* 12:e1004768. doi: 10.1371/journal.pcbi.1004768
- Yang, S., Wang, J., and Deng, B. (2020). Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 148–162. doi: 10.1109/tnnls.2019.2899936
- Yang, S., Wang, J., and Li, S. (2015). Cost-efficient FPGA implementation of basal ganglia and their Parkinsonian analysis. *Neural Netw.* 71, 62–75. doi: 10.1016/j.neunet.2015.07.017
- Yang, S., Wang, J., Hao, X., Li, H., Wei, X., Deng, B., et al. (2021). BiCoSS: Toward large-scale cognition brain with multigranular neuromorphic architecture. *IEEE Trans. Neural Netw. Learn. Syst.* [Epub ahead of print]. doi: 10.1109/TNNLS.2020.3045492
- Yang, S., Wang, J., Deng, B., Liu, C., Li, H., and Fietkiewicz, C. (2018). Real-time neuromorphic system for large-scale conductance-based spiking neural networks. *IEEE Trans. Cybern.* 49, 2490–2503. doi: 10.1109/tcyb.2018.2823730
- 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
- Zhou, F., Zhou, Z., Chen, J., Choy, T. H., and Chai, Y. (2019). Optoelectronic resistive random access memory for neuromorphic vision sensors. *Nat. Nanotechnol.* 14, 776–782. doi: 10.1038/s41565-019-0501-3

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 Yang, Gao, Wang, Deng, Lansdell and Linares-Barranco. 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 Analog Implementation of Neural Engineering Framework-Inspired Spiking Neuron for High-Dimensional Representation

Avi Hazan and Elishai Ezra Tsur*

Neuro-Biomorphic Engineering Lab, Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Shuangming Yang,
Tianjin University, China
Qinru Qiu,
Syracuse University, United States

*Correspondence:

Elishai Ezra Tsur
elishai@NBEL-lab.com

Specialty section:

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

Received: 08 November 2020

Accepted: 25 January 2021

Published: 22 February 2021

Citation:

Hazan A and Ezra Tsur E (2021)
Neuromorphic Analog Implementation
of Neural Engineering
Framework-Inspired Spiking Neuron
for High-Dimensional Representation.
Front. Neurosci. 15:627221.
doi: 10.3389/fnins.2021.627221

Brain-inspired hardware designs realize neural principles in electronics to provide high-performing, energy-efficient frameworks for artificial intelligence. The Neural Engineering Framework (NEF) brings forth a theoretical framework for representing high-dimensional mathematical constructs with spiking neurons to implement functional large-scale neural networks. Here, we present OZ, a programmable analog implementation of NEF-inspired spiking neurons. OZ neurons can be dynamically programmed to feature varying high-dimensional response curves with positive and negative encoders for a neuromorphic distributed representation of normalized input data. Our hardware design demonstrates full correspondence with NEF across firing rates, encoding vectors, and intercepts. OZ neurons can be independently configured in real-time to allow efficient spanning of a representation space, thus using fewer neurons and therefore less power for neuromorphic data representation.

Keywords: neural engineering framework, spiking neural networks, neuromorphic electronics, neuromorphic engineering, brain-inspired electronics

INTRODUCTION

Albeit artificial intelligence has emerged as the focal point for countless state-of-the-art developments, in many ways, it is nullified when compared with biological intelligence, particularly in terms of energy efficiency. For instance, the honeybee is capable of exceptional navigation while possessing just under 1 million neurons and consuming only 10^{-3} W of power. Comparably, an autonomous car would need to utilize over a 10^3 W of sensing and computing power, demonstrating lamentable energetic efficiency decreased a millionfold (Liu et al., 2014). Consequentially, brain-inspired hardware designs have been used in numerous applications, particularly in neuro-robotics (Krichmar and Wagatsuma, 2011; Zaidel et al., 2021) and smart-edge devices (Krestinskaya et al., 2019; Zhang et al., 2020). In neuromorphic computing architectures, the computational principles of biological neural circuits are utilized to design artificial neural systems. A neuromorphic circuit

comprises densely connected, physically implemented computing elements (e.g., silicon neurons), which communicate with spikes (Tsur and Rivlin-Etzion, 2020). Notable neuromorphic hardware includes the TrueNorth (DeBole et al., 2019), developed by IBM research, the Loihi (Davies et al., 2018), developed by Intel Labs, the NeuroGrid (Benjamin et al., 2014), developed at Stanford University, and the SpiNNaker (Furber et al., 2014), developed at the University of Manchester. One theoretical framework, which allows for efficient data encoding and decoding with spiking neurons, is the Neural Engineering Framework (NEF) (Eliasmith and Anderson, 2003). NEF is one of the most utilized theoretical frameworks in neuromorphic computing. It was adopted for various neuromorphic tasks, ranging from neuro-robotics (DeWolf et al., 2020) to high-level cognition (Eliasmith et al., 2012). It was compiled to work on multiple neuromorphic hardware using Nengo, a Python-based “neural compiler,” which translates high-level descriptions to low-level neural models (Bekolay et al., 2014). NEF was shown to be incredibly versatile, as a version of it was compiled on each of the neuromorphic hardware designs listed earlier (Mundy et al., 2015; Boahen, 2017; Fischl et al., 2018; Lin et al., 2018), although they do not follow the same paradigm of neuromorphic implementation. Although the Loihi, the TrueNorth, and the Spinnaker are pure digital systems, in the sense that both computing and communication are held digitally, the NeuroGrid is a mixed analog–digital circuit. In the Neurogrid, synaptic computations were implemented with analog circuitry. Although these general-purpose computing architectures adopted the digital realm for better adherence with application programming and ease of fabrication, analog implementation of synapses (such as the one implemented in the NeuroGrid) is commonly found in analog neuromorphic sensing and signal processing. Notably, some of the first and most significant successes in neuromorphic architectures have been in vision (Indiveri and Douglas, 2000) and sound (Liu and Delbruck, 2010) processing.

NEF-inspired neurons were previously directly implemented in both digital and analog circuitry. For example, NEF-inspired neurons were implemented on a digital Field-Programmable Gate Array (FPGA)-circuit and used for pattern recognition (Wang et al., 2017). However, it is not clear if such implementations can approximate the density, energy efficiency, and resilience of large-scale neuromorphic systems (Indiveri et al., 2011). Current analog implementations of NEF-inspired neurons rely on the circuit fabrication’s stochasticity to constitute the variational activity patterns required to span a representation space. The activity pattern of these neurons cannot be modulated or programmed, and therefore, using them for precise representation of a mathematical construct—even in low dimension—requires a large number of neurons and, hence, has suboptimal energy consumption (see section “Discussion” for further details) (Mayr et al., 2014; Boahen, 2017). Here, we present OZ, a programmable, analog implementation of NEF-inspired spiking neuron. OZ utilizes several of the most well-known building blocks for analog spiking neurons to provide a design with a programmable high-dimensional response curve and a temporally integrated output.

MATERIALS AND METHODS

Circuit Simulations and Analysis

All circuit simulations in this study were executed using LTspice, offered by Analog Devices (2008). The simulator is based on the open-sourced SPICE framework (Nagel and Pederson, 1973), which utilizes the numerical Newton–Raphson method to analyze non-linear systems (Nichols et al., 1994). Signal analysis was performed using the Python scripts we developed. Curve and surface fittings were performed using MATLAB’s curve fitting toolbox. Simulation files are available upon request.

Distributed Neuronal Representation With Neural Engineering Framework

Let a be a representation, or a function, of a stimulus x using $a = f(x)$. With NEF, high-level network specifications, given in terms of vectors and functions, are transformed to a set, or an ensemble, of spiking neurons. A neural representation will therefore take the form of $a = G(J(x))$, where G is a spiking neuron model [e.g., the leaky-integrate-and-fire (LIF) model (Burkitt, 2006)] and J is the integrated inputs introduced to the neuron. NEF uses a distributed neuron representation, where each neuron i responds independently to x , resulting in $a_i = G_i(J_i(x))$. One possible modeling for J would be $J = \alpha x J^{bias}$, where α is a gain term and J^{bias} is a fixed background current. Neurons often have some preferred stimuli e (preferred direction, or encoder) to which they respond with a high frequency of spikes [e.g., direction selectivity in retinal ganglion cells (Ankri et al., 2020)]. J will therefore be more appropriately defined using: $J = \alpha x \cdot e J^{bias}$, where $x \cdot e$ equals 1 when both x and e are in the same direction, and 0 when they are opposing each other. To conclude, in NEF, a neuron firing rate δ_i is defined using:

$$\delta_i(x) = G_i[\alpha_i e_i x J_i^{bias}] \quad (1)$$

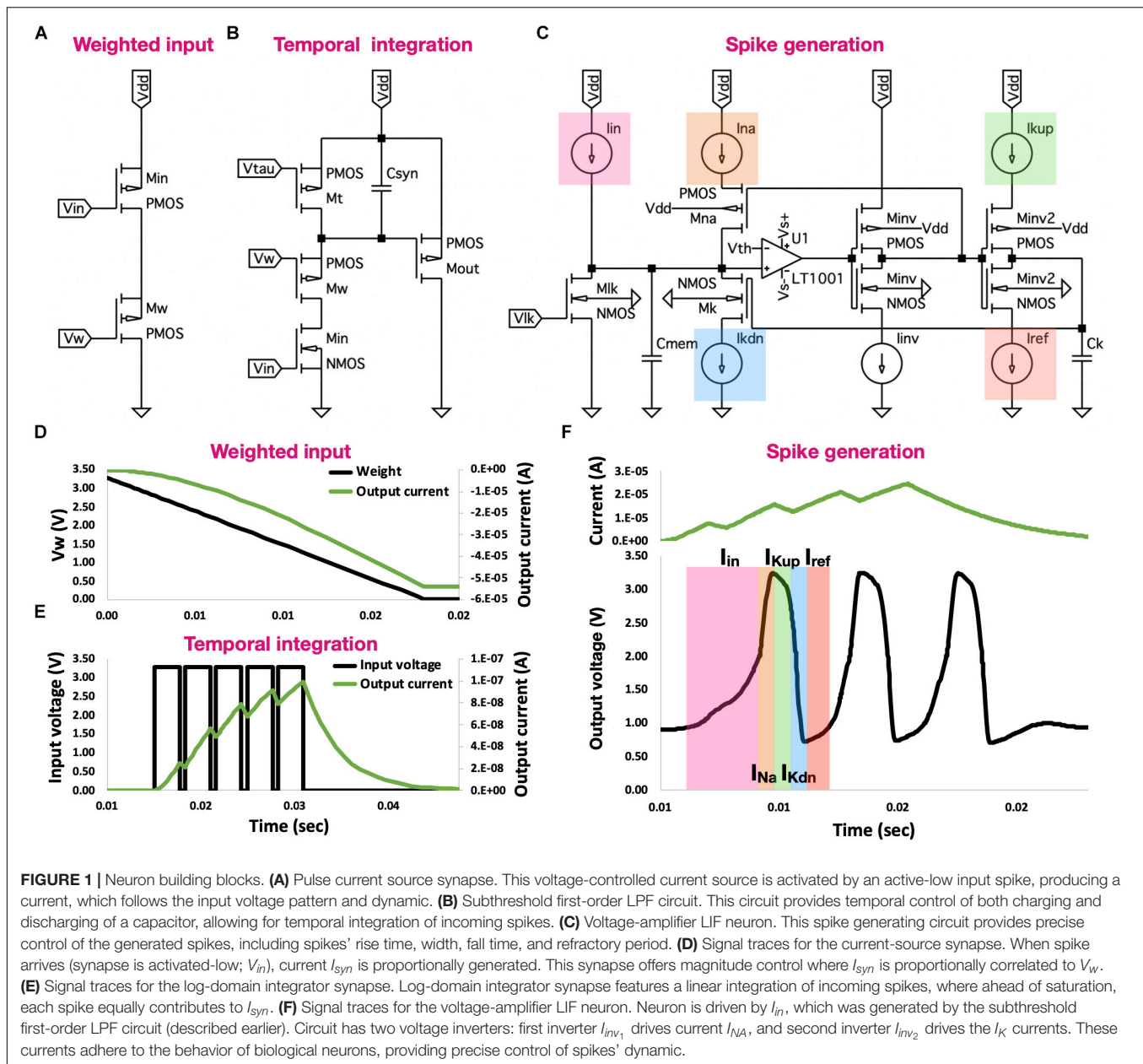
An ensemble of neurons in which each neuron has a gain and preferred direction distributively represents a vectorized (or high-dimensional) stimulus x . The represented stimuli \hat{x} can be decoded using:

$$\hat{x} = \sum_i a_i * h d_i \quad (2)$$

Where d_i is a linear decoder, which was optimized to reproduce x using least squared optimization and $a_i^* h$ is a spiking activity a_i , convolved with a filter h (both are functions of time). NEF is described in detail in Eliasmith and Anderson (2003) and succinctly reviewed in Stewart and Eliasmith (2014). NEF is the foundation upon which our neuron model is built. Particularly, it is utilized here to represent a high-dimensional stimulus with spiking neurons distributively.

Analog Building Blocks

In a seminal review by Indiveri et al. (2011) “Neuromorphic silicon neuron circuits,” the fundamental building blocks of analog neurons were described. Among them were (1) the



pulse current source synaptic circuit, (2) the subthreshold first-order LFP circuit, and (3) the voltage-amplifier LIF neuron (Figures 1A–C). We will briefly revisit these circuits here, as they constitute the OZ neuron's main building blocks.

The pulse current-source synapse (Figure 1A), proposed by Mead (1989), was one of the first artificial synapse circuits created. It is a voltage-controlled current source, which is driven by an active-low input spike. The resulting current I_{syn} is defined using:

$$I_{syn} = I_0 e^{-\frac{\kappa}{U_T}(V_w - V_{dd})} \quad (3)$$

Where V_{dd} is the supply voltage, I_0 is the leakage current of the transistor M_w , which is activated in the subthreshold regime, κ is the subthreshold slope factor and U_T is a thermal voltage (at

room temperature, it is approximately 26 mV). This circuit allows for controlling the magnitude of I_{syn} such that when V_w equals V_{dd} , I_{syn} is I_0 . As we decrease V_w , I_0 is scaled up exponentially, increasing I_{syn} accordingly (Figure 1D). While offering control over I_{syn} 's magnitude, the pulse current-source synapse does not provide temporal modulation.

The subthreshold first-order LFP circuit (Figure 1B), proposed by Merolla and Boahen (2004), offers linear integration of incoming spikes. This circuit is built upon the charge and discharge synapse [described in Bartolozzi and Indiveri (2007)], which provides temporal control of charging and discharging of C_{syn} . In the charge and discharge synapses, the incoming active-high spikes activate the transistor M_{in} . During a spike, V_{syn} decreases linearly, at a rate set by the net current $I_w - I_t$, where

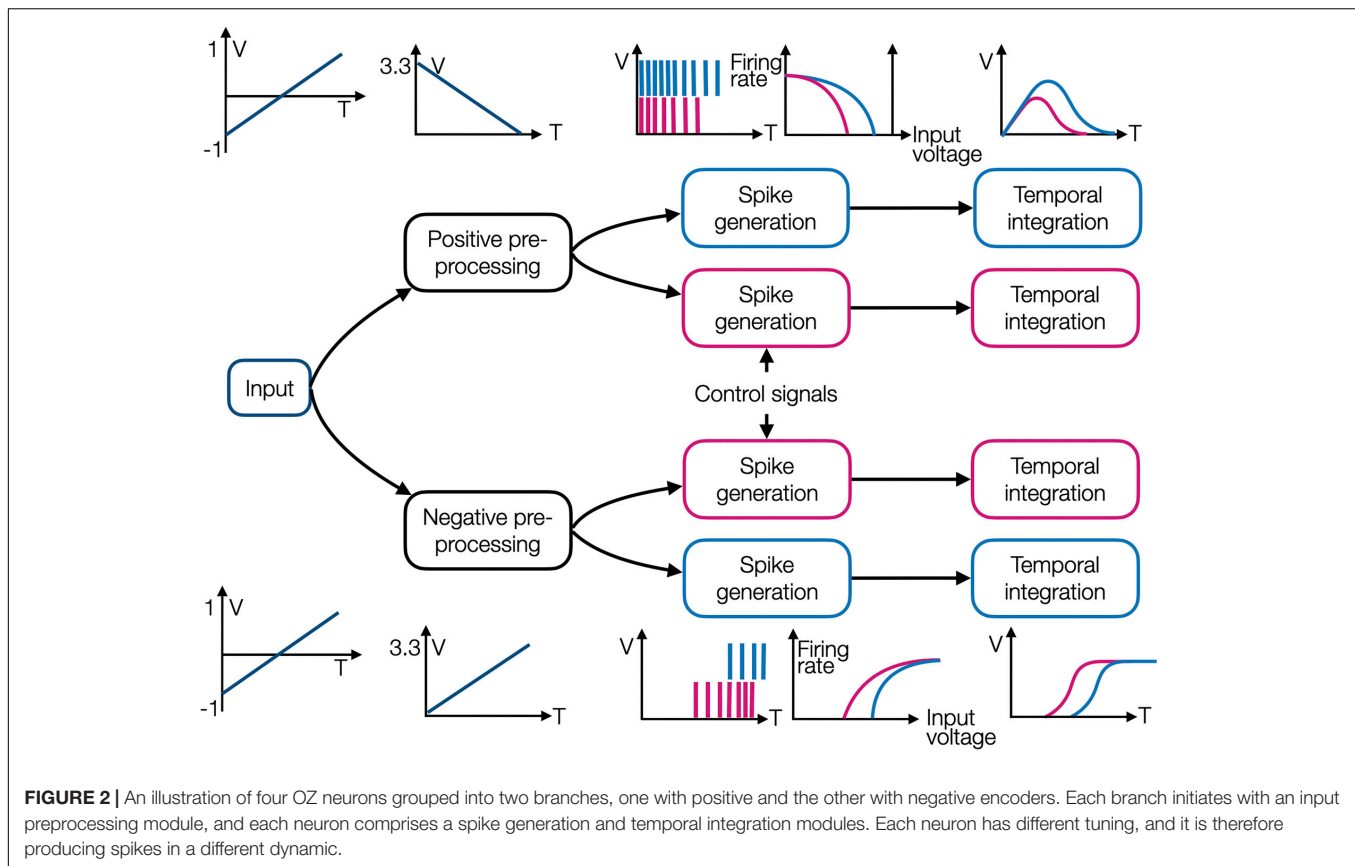


FIGURE 2 | An illustration of four OZ neurons grouped into two branches, one with positive and the other with negative encoders. Each branch initiates with an input preprocessing module, and each neuron comprises a spike generation and temporal integration modules. Each neuron has different tuning, and it is therefore producing spikes in a different dynamic.

I_w is the current driven through transistor M_w (and regulated by V_w) and I_τ is the current driven through transistor M_τ (and regulated by V_τ). This net current is responsible for discharging C_{syn} . The linearly decreasing V_{syn} drives I_{syn} by regulating transistor M_{out} . In this log-domain circuit, the logarithmic relationship between the transistor's V_{gs} and its current is used to exhibit overall linear properties [see (Indiveri et al., 2011) for a detailed analysis]. The governing equations of this synapse during a spike I_{syn}^{spike} and between spikes I_{syn}^{flat} are:

$$I_{syn}^{spike} = \frac{I_0}{I_\tau} \left(1 - e^{-\frac{(t-t_i^-)}{\tau_c}} \right) I_{syn}^- e^{-\frac{(t-t_i^-)}{\tau_c}} \quad (4)$$

$$I_{syn}^{flat} = I_{syn}^- e^{-\frac{(t-t_i^+)}{\tau_d}} \quad (5)$$

where t_i^- and t_i are the times at which spike i arrives and terminates, respectively, I_{syn}^- and I_{syn} are the I_{syn} in times t_i^- and t_i , respectively, τ_c is the time constant for the capacitor charge, which equals $U_T = C/\kappa (I_w - I_\tau)$, and τ_d is the time constant for the capacitor-discharge, which equals $U_T = C/\kappa I_\tau$. Controlling the charge and discharge of C_{syn} allows for temporal control of both rise and fall times of V_{syn} , thus providing the ability to temporally integrate multiple incoming spikes (Figure 2E).

The voltage-amplifier LIF neuron is a spike generating circuit proposed by van Schaik (2001) and Figure 1C. This circuit enhances the classic axon-hillock neuron design [described in

Mead (1989)] with precise control of the generated spikes' dynamic, including spikes' rise time, width, fall time, and refractory period. Capacitor C_{mem} models the neuron membrane and V_{lk} , which regulates the conductance of transistor M_{lk} , controls its leakage current I_{lk} . In the absence of an input current from incoming spikes (flat phase), I_{lk} drives the membrane voltage V_{mem} down to 0 V. When an input current is apparent, the net incoming current $I_{in} - I_{lk}$ is charging C_{mem} , increasing V_{mem} . When V_{mem} exceeds V_{th} , an action potential is generated via an operational amplifier (op-amp). This action potential is introduced into a voltage inverter, where high logical states are transformed into low logical states and vice versa. A low logical voltage state activates transistor M_{Na} , through which I_{Na} current is driven, charging C_{mem} and creating a sustained high voltage (constituting the spike). A second voltage inverter drives I_{kup} through transistor M_{inv2} , charging C_k , thus controlling spike's width. As C_k is charging, it activates transistor M_k , through which I_k is driven. I_k discharges C_{mem} and when V_{mem} drops below V_{th} , the amplifier's output drops to a low state. In response, the first voltage inverter's output is driven high, deactivating transistor M_{Na} , thus terminating I_{Na} . The second inverter's output voltage is driven low, terminating I_{kup} and allowing I_{ref} to discharge C_k . As long as I_{ref} is not strong enough to discharge C_k , the circuit cannot be further stimulated by incoming current (assuming $I_{in} < I_k$), constituting a refractory period. The generated spikes are shown in Figure 1F. This process is a direct embodiment of the biological behavior, in which an influx of sodium ions

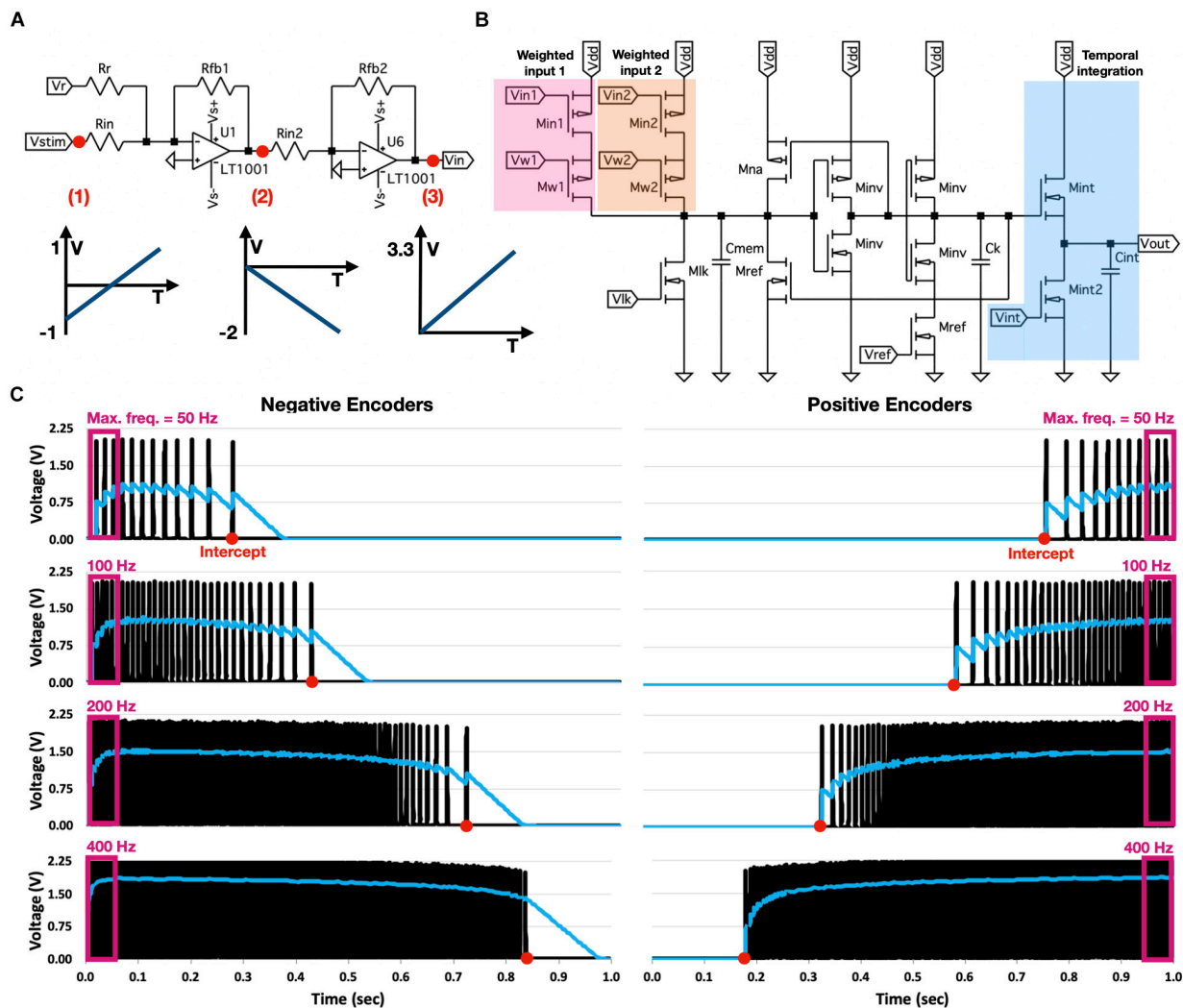


FIGURE 3 | OZ neuron analog design. **(A)** Preprocessing module for negatively encoded OZ neurons. Module inverses input voltage, aligns it to initiate at 0 V, and reinverts and scales it to terminate at 3.3 V. **(B)** Neuron's spike generator. Voltages from two weighted inputs are transformed into a proportional current, injected into a modified voltage-amplifier LIF neuron. Neuron produces a spike train according to its response dynamic. Spike train is introduced into a temporal integration circuit. **(C)** Eight OZ neurons, four of them are positively encoded (right), and four are negatively encoded (left). All neurons were stimulated with a linearly increasing voltage, rising from -1 to 1 V. Each neuron was modulated with various values of V_{ik} to produce a spike train at a particular rate, starting from a specific input (intercept).

(Na^+) and a delayed outflux of potassium ions (K^+) govern the initiation of an action potential.

RESULTS

Circuit Design

In our circuit design, stimulus x is introduced through preprocessing modules to two branches, one connected to positively encoded OZ neurons and the other to negatively encoded OZ neurons. These preprocessing modules accept an input voltage ranging from -1 to 1 V (corresponding to the default input normalization scheme taken by NEF) and produce an output voltage ranging from 0 to 3.3 V. Each OZ neuron is comprised of two consecutive modules: a spike generator and

a temporal integrator. Each spike generator is characterized by a tuning curve, modulated using control signals, thus realizing Eq. 1. A generated spike train is introduced to a temporal integrator, which integrates the incoming spikes, thus realizing Eq. 2 and constituting a NEF-inspired neuron. The circuit schematic for two negatively encoded and two positively encoded neurons is shown in Figure 2. The negative preprocessing circuit comprises two consecutive modules: the first one inverses the voltage and aligns it to initiate at 0 V, and the second reinverts and scales it so it will terminate at 3.3 V (circuit's V_{dd}) (Figure 3A). The first module uses an op-amp based adder to add 1 V to the input signal (aligning it to 0 V) and inverts it according to $V_o = -(V V^-)$, where V and V^- are the op-amp's input terminals. The resulted voltage is ranging from 0 to 2 V. The second module uses an inverter amplifier that

scales its input voltage according to $-R_{fb}/R_{in}$, where R_{fb} is the feedback resistor and R_{in} is the amplifier's input terminal resistor. Here, $R_{fb2} = 1.65 \text{ k}\Omega$ and $R_{in2} = 1 \text{ k}\Omega$, achieving a scaling factor of -1.65 , which transform 2 to 3.3 V output. The positive preprocessing module resembles the negative preprocessing module, with the addition of another voltage inverter, which produces a similar waveform, initiating at 3.3 V and terminating at 0 V.

The OZ neuron is shown in **Figure 3B**. It is based on modified versions of the pulse current source synaptic circuit (for weighted input), the voltage-amplifier LIF neuron (for spike generation), and the subthreshold first-order LPF circuit (for temporal integration). The pulse-current source synapse is used to convert an input voltage to a proportional current, introduced into the spike generation circuit, and defined by V_w according to Eq. 3. The voltage-amplifier LIF neuron's response dynamic is predominantly determined by the values of I_{kup} , I_{kdn} , I_{Na} , and I_{ref} and the leakage current I_{lk} (driven through transistor M_{lk}), which are regulated, respectively, by V_{kup} , V_{kdn} , V_{Na} , V_{ref} , and V_{lk} via dedicated transistors. Therefore, this neuron has five degrees of freedom (DOF): V_{lk} controls the discharge rate of C_{mem} , V_{ref} controls spikes' refractory period, V_{kup} and V_{kdn} control the fall time of the generated spikes, and V_{Na} controls the spikes' rise time. Furthermore, its spiking dynamic relies on an op-amp, which is comprised of multiple transistors and resistors. OZ's spike generator is a NEF-optimized circuit design, where V_{up} , V_{Na} , and V_{kdn} are redundant. Furthermore, it does not rely on op-amp for spike generation, as the amplifier has no significant functional effect in terms of neuron's firing rate and intercept (see section "Discussion"). A NEF-tailored design should also enable high-dimensional input representation, which can be achieved by concatenating the input module (highlighted in **Figure 3B** as weighted input; see section "Circuit Analysis"). Finally, temporal integration can be achieved via a simplified LPF temporal integration circuit. In OZ, capacitor C_{int} is charged by current I_{int} , which is activated by the generated spike train and driven through transistor M_{int} . C_{int} is discharging at a constant rate through a leakage current, which is driven through transistor M_{int2} and regulated by a continuously held voltage V_{int} . The voltage on C_{int} constitutes the OZ neuron's output.

A useful way of representing a neuron's response to varying inputs is by using a response, or a tuning curve, which is one of the most fundamental concepts of NEF. In NEF, a tuning curve is defined using an intercept, the value for which the neuron starts to produce spikes at a high rate, and its maximal firing rate. OZ's tuning curve can be programmed to control both. For circuit analysis, we built eight OZ neurons, four with positive and four with negative encoders. Each neuron has $d + 2$ DOF, where d is the dimensionality of the input, corresponding to d values of V_w , which regulate each input dimension, and V_{lk} and V_{ref} correspond to the two other DOF. To demonstrate OZ, we built eight neurons; each was defined to feature a different intercept and maximal firing rate. Each neuron was stimulated with the same input voltage, which linearly increased from -1 to 1 V over 1 s. Results are shown in **Figure 3C**.

Circuit Analysis

Architectural Design

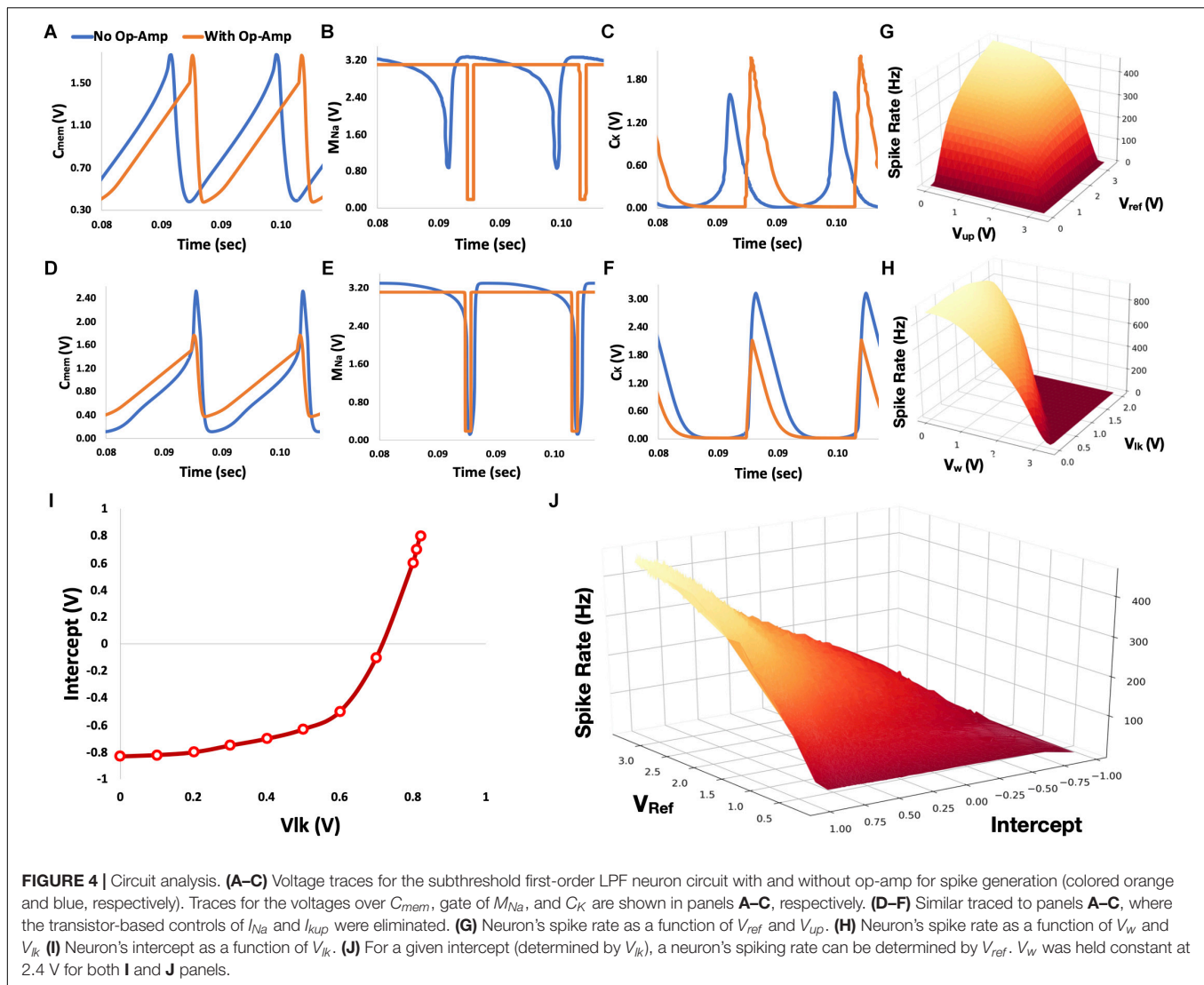
For the sake of discussion, we will consider one-dimensional (1D) OZ neurons. First, we shall consider the classic voltage-amplifier LIF neuron, shown in **Figure 1C**. This design relies on an op-amp for spike generation. From a functional perspective, the op-amp provides the neuron with a digital attribute, splitting the neuron into an analog pre-op-amp circuit and a digital post-op-amp circuit. Particularly, when an incoming current is inducing V_{mem} to exceed a predefined threshold voltage, the op-amp yields a square signal, which generates a sharp I_{Na} response. This fast response induces sharp swing-ups in V_{mem} and V_{out} . Without the op-amp, this transition between states is gradual (**Figures 4A–C**). Although both designs permit spike generation, the op-amp-based design can generate spikes in a higher frequency and amplitude. To compensate for that, we can discard both I_{Na} and I_{kup} controls through the removal of their regulating transistors. Removing these resistance-inducing transistors maximizes I_{Na} and I_{kup} , thus achieving op-amp-like frequency and amplitude (**Figures 4D–F**). Moreover, without the op-amp, there is no need to explicitly define a threshold, providing a more straightforward and biologically plausible design.

Neuron Control

Did we lose control over the maximal firing rate over our neuron by eliminating the regulation of I_{kup} ? Fortunately, both I_{kup} and I_{ref} impact neuron's firing rate. Although I_{kup} limits neuron's firing rate by governing the rise time of the generated spikes, I_{ref} does that by setting the refractory period between spikes. Controlling both currents is redundant as both imply similar constraints, as shown in **Figure 4G** (V_{lk} and V_w are held constant at 2.2 and 0.5 V, respectively).

V_{lk} controls the discharge rate of C_{mem} by regulating a leakage current through transistor M_{lk} . As long as this leakage current is lower than the input current (driven through the weighted input module), V_{mem} will rise toward saturation. As we decrease V_{lk} , leakage current drops and C_{mem} charged faster. As a result, the neuron's intercept (the input value for which the neuron's initiate spikes at a high rate) increases for positively encoded neurons and decreases for negatively encoded neurons. Neurons exhibit maximal firing rate when their input voltage is either -1 or 1 V, depending on the neuron's encodings. The maximal firing rate is proportionally dependent on the charging status of membrane capacitance C_{mem} . The faster C_{mem} is charging, the more frequent the neuron will emit spikes. However, a neuron's maximal firing rate is not entirely decoupled from its intercept. Although a neuron's intercept is controlled by V_{lk} , it can also be modulated by V_w , which provides a magnitude control for the input current. Therefore, although V_{lk} can be used to define the neuron's intercept, V_w can impose on it a firing rate constraint. For example, the neuron's spiking rate will not exceed 400 Hz, when V_w is set to 2.2 V. V_{lk} and V_w imposed constraint on neuron's spiking rate is demonstrated in **Figure 4H** (V_{ref} and V_{up} are held constant at 3.3 and 1 V, respectively).

Figures 4I,J summarizes the control of the neuron's intercept and spiking rate using V_{ref} and V_{lk} (whereas V_w was held constant at 2.4V). Through curve and surface fittings



($R^2 > 0.98$), neuron's intercept N_{int} can be described with:

$$N_{int}(V_{lk}) = 5.611 \cdot 10^{-3} e^{6.911 \cdot V_{lk}} - 0.8178 \quad (6)$$

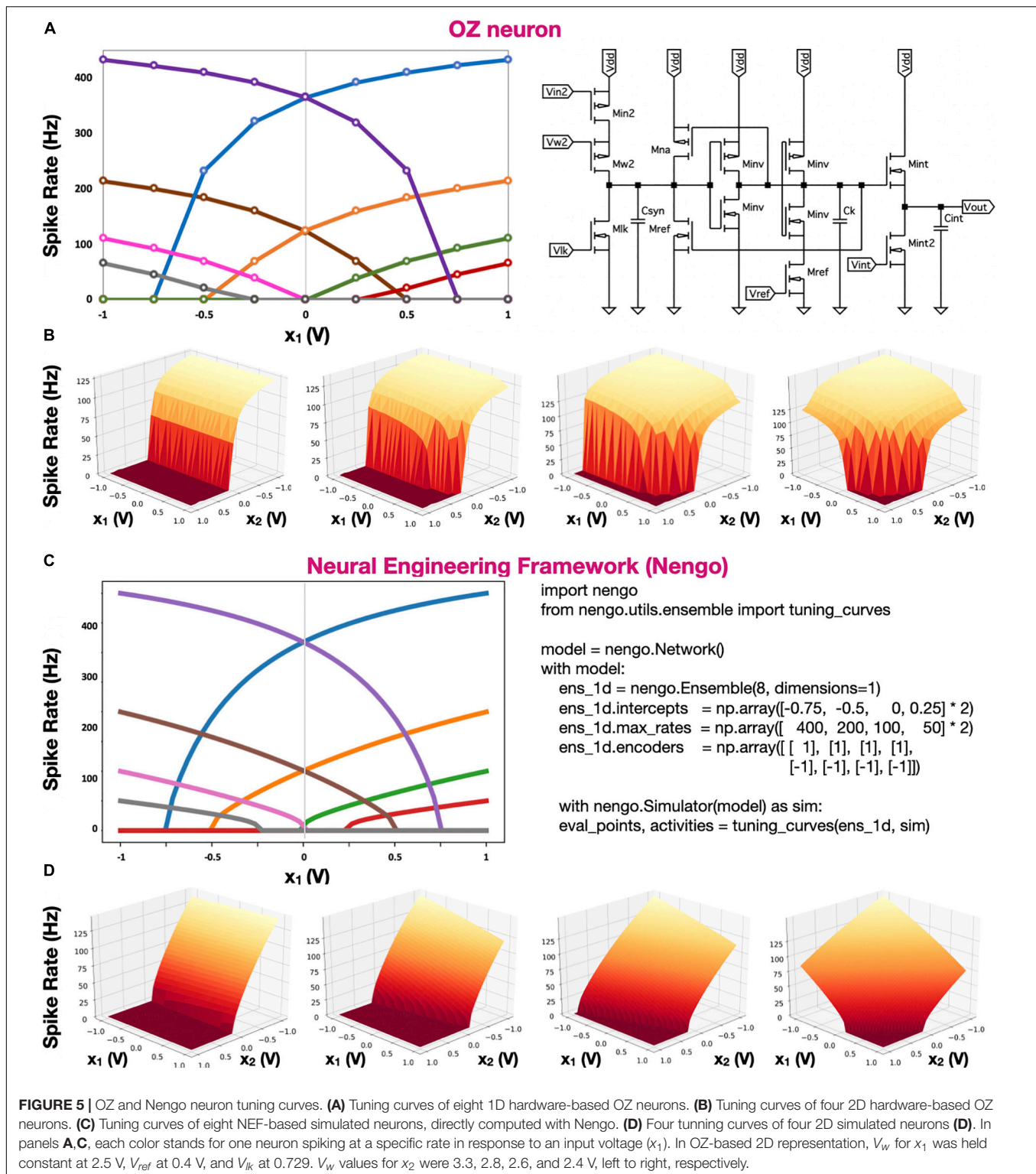
and neuron's maximal firing rate N_{FR} with:

$$N_{FR}(V_{ref}, N_{int}) = -36180 V_{ref} 55 N_{int} - 34.4 V_{ref}^2 66.3 V_{ref} N_{int} \quad (7)$$

In **Figure 5**, the tuning curves of our eight OZ neurons, along with the tuning curves of eight simulated neurons, which were computed directly with NEF, are demonstrated. The tuning curves indicate varying intercepts and spiking rates, showcasing the produced spike trains' high predictability and the full correspondence between our hardware design and NEF. In **Figures 5B,D**, we compared 2D tuning curves. It was achieved with OZ by concatenating two weighted inputs: x_1 and x_2 , weighting them with V_{w1} and V_{w2} , respectively. Results show the high predictability of the neuron in response to a high-dimensional stimulus.

DISCUSSION

Numerous digital and analog designs of spiking neurons have been previously proposed. For example, Yang et al. (2018b) proposed a biologically plausible, conductance-based implementation of spiking neurons with an FPGA. This design was used to simulate 1 million neurons by utilizing six state-of-the-art FPGA chips simultaneously, achieving biological plausibility and scale (Yang et al., 2018a). Furthermore, it was shown to feature multicompartamental neuron design, supporting the morphologically detailed realization of neural networks (Yang et al., 2019). Biologically plausible spiking neurons were also implemented in analog circuits, featuring spike adaptation (Aamir et al., 2017). Although incredibly versatile and highly configurable, these designs were guided by a bottom-up approach, tailored to reproduce biological behavior. However, to achieve function-optimized neural networks (e.g., for neurorobotics or other smart-edge devices), top-bottom modeling is more suitable



(Eliasmith and Trujillo, 2014). By throwing out morphological and physiological constraints, the NEF allows top-down optimization, with which high-level specification can be realized in spiking neurons with a minimal number of explicitly defined neuronal characteristics.

NEF is one of the most utilized theoretical frameworks in neuromorphic computing. A version of NEF was compiled on various neuromorphic digital systems, such as Intel's Loihi and IBM's TrueNorth (Fischl et al., 2018; Lin et al., 2018), as well as on hybrid analog/digital systems such as the

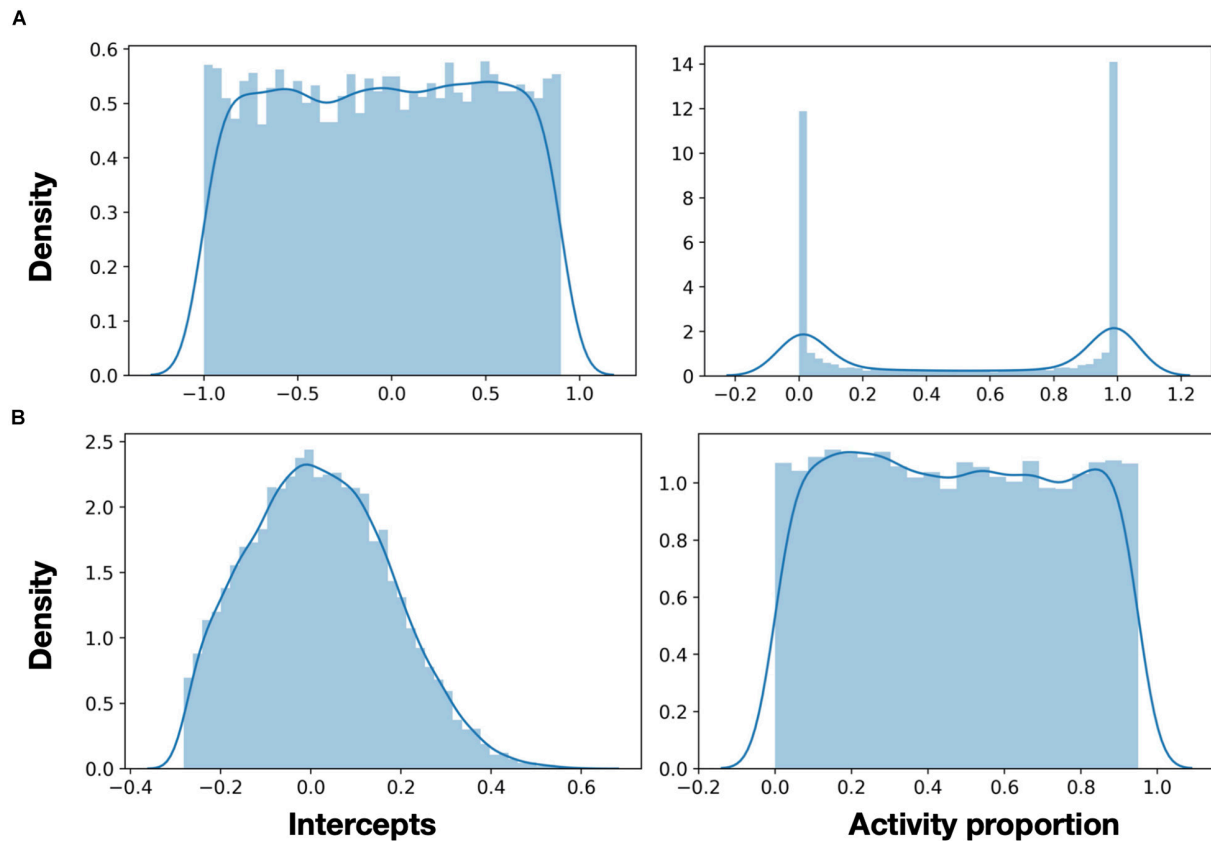


FIGURE 6 | Neurons activity in high dimension. **(A)** In 32D representation, a uniform distribution of intercepts (left) creates many neurons, which are either always or never active (right). **(B)** Using a rational distribution of intercepts (using Eq. 7), a uniform activity pattern of neurons across the representation space can be generated.

NeuroGrid (Boahen, 2017). NEF-inspired neurons were directly implemented in both digital (Wang et al., 2017) and analog (Indiveri et al., 2011) circuitry. Although digital NEF-inspired implementations are versatile and programmable, they are fundamentally less energy-efficient and footprint-restricted in comparison with analog circuitry (Amara et al., 2006). Current analog implementations of NEF-inspired neurons rely on the inherent stochasticity in the fabrication process of integrated circuits to create the variational neurons' tuning required to span a representation space (Mayr et al., 2014; Boahen, 2017) or to support machine learning (Tripathi et al., 2019).

Neurons in NEF represent mathematical constructs, and their accuracy of that representation is fundamentally limited to the neurons' tuning curves. A tuning curve is defined using an intercept and maximal spike rate. Intercepts represent the part of the representation space for which the neuron will fire. In 1D, uniformly distributed intercepts will uniformly span the representation space. A neuron with an intercept of 0 will be active for 50% of that space, and a neuron with an intercept of 0.75 will be active for only 7.5% of that space. However, using randomly distributed tuning curves would require many more neurons to achieve adequate space spanning. When an input does not invoke a neuron to spike, that neuron is essentially a waste of space and energy.

Moreover, as we advance toward representing values in higher dimensions, articulating and carefully defining neurons' tuning curves become a critical design factor. This design factor was attested by the authors of Mayr et al. (2014), as they discussed their analog implementation of a NEF-inspired neuron: "as the spread of the curves is determined by random effects of the manufacturing process, individual instances of the ADC [the designated application for that design] have to be checked for sufficient spread, thus defining a yield in terms of ADC resolution. When comparing the two families of tuning curves, the main observation is that the Nengo generated neurons tend to vary more, especially in their gain... this has a significant impact on the overall computation. If the neurons do not encode for sufficiently different features of the input signal, the representation of the input signal degrades" (Mayr et al., 2014).

Moreover, our proposed implementation offers a high-dimensional representation. Distributing intercepts uniformly between -1 and 1 makes sense for 1D ensembles. Because a neuron's intercept defines the part of the representation space in which this neuron is firing, in 1D representation, uniformly distributed intercepts create a uniform spanning of that space. In higher dimensions, the proportions of activity are getting smaller (or larger for negatively encoded neurons). In high dimensions, the naive distribution of intercepts results in many neurons,

either rarely producing spikes or always active (**Figure 6A**). In both cases, these neurons are essentially not contributing to the representation. A representation space in 2D is a 3D sphere, in which each neuron's encoder points to a cap, which specifies the space in which that neuron is active (Gosmann and Eliasmith, 2016). The intercept is the location of the cap's cutoff plane. The ratio between the cap's and the sphere's volumes is the percentage of the representation space in which a neuron is active. A generalized sphere in a higher dimension is a hyper-sphere. The volume of a hyper-sphere cap v_{cap} is defined with:

$$v_{cap} = \frac{1}{2} C_d r^d I_{2rh-h^2/r^2} \left(\frac{d+1}{2}, \frac{1}{2} \right) \quad (8)$$

where C_d is the volume of a unit hypersphere of dimension d and radius r , h is the cap's height, and $I_x(a, b)$ is the regularized incomplete beta function. Here, $r = 1$ (representation is in $[-1, 1]$), and $h = x - 1$, where x is the intercept. The ratio p between the hypersphere's volume C_d and its cap's volume v_{cap} is:

$$p = \frac{1}{2} I_{1-x^2} \left(\frac{d+1}{2}, \frac{1}{2} \right) \quad (9)$$

To more efficiently span in high-dimensional representation space, we can use the inverse of Eq. 6 to derive a desired p value, the intercept, which will create it. This equation is defined with:

$$x = \sqrt{1 - I_{2p}^{-1} \left(\frac{d+1}{2}, \frac{1}{2} \right)} \quad (10)$$

With Eq. 7, we can generate the intersects to better span the representation space. Utilizing this equation can provide the intersects distribution for which the spikes activity pattern is uniform (**Figure 6B**). This is a clear example of the importance of being able to modulate neuron's tuning curves in high-dimensional representation. The importance of the discussion earlier was recently highlighted in DeWolf et al. (2020) in the context of neuro-robotics.

Here, we presented the OZ neuron—a programmable analog implementation of a spiking neuron, which can have its tuning curve explicitly defined. With our system design, for

a uniform distribution of tuning curves (required in most low-dimensional applications), only one among the positive and negative branches has to be defined, cutting in half the number of neurons, which have to be controlled. Because we can design the neurons' tuning curve to accurately span the representation space following a particular application's needs, the required number of neurons for spanning that space can be significantly reduced. Moreover, uniquely, neurons' tuning curves can be changed in real-time to provide dynamically modulated neuromorphic representation. However, when the required number of neurons is large, the apparent overhead of control must be considered. Our design can be scaled to a full very large-scale integration neuromorphic circuit design, providing analog, distributed, and energy-efficient neuromorphic representation of high-dimensional mathematical constructs.

DATA AVAILABILITY STATEMENT

Model simulation will be provided upon request.

AUTHOR CONTRIBUTIONS

AH designed the circuits and performed circuit simulation and analysis. EE conceptualized the research, designed the circuits, and wrote the manuscript. Both authors contributed to the article and approved the submitted version.

FUNDING

This research was supported by the Israel Innovation Authority (EzerTech) and the Open University of Israel research grant.

ACKNOWLEDGMENTS

The authors would like to thank Tamara Perelman Tsur for her insightful comments.

REFERENCES

- Aamir, S. A., Muller, P., Kriener, L., Kiene, G., Schemmel, J., and Meier, K. (2017). "From LIF to AdEx neuron models: accelerated analog 65 nm CMOS implementation," in *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Turin. doi: 10.1109/BIOCAS.2017.8325167
- Amara, A., Amiel, F., and Ea, T. (2006). FPGA vs. ASIC for low power applications. *Microelectron. J.* 37, 669–677. doi: 10.1016/j.mejo.2005.11.003
- Analog Devices (2008). *LTspice simulator*. Available online at: <http://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html> (accessed September 26, 2020).
- Ankri, L., Ezra-Tsur, E., Maimon, S. R., Kaushansky, N., and Rivlin-Etzion, M. (2020). Antagonistic center-surround mechanisms for direction selectivity in the retina. *Cell Rep.* 31:107608. doi: 10.1016/j.celrep.2020.107608
- Bartolozzi, C., and Indiveri, G. (2007). Synaptic dynamics in analog VLSI. *Neural Comput.* 19, 2581–2603. doi: 10.1162/neco.2007.19.10.2581
- Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform.* 7:48. doi: 10.3389/fninf.2013.00048
- Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A., 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
- Boahen, K. (2017). A neuromorph's prospectus. *Comput. Sci. Eng.* 19, 14–28. doi: 10.1109/MCSE.2017.33
- Burkitt, A. (2006). A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biol. Cybern.* 95, 1–19. doi: 10.1007/s00422-006-0068-6
- 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
- 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

- DeWolf, T., Jaworski, P., and Eliasmith, C. (2020). Nengo and low-power AI hardware for robust, embedded neurorobotics. *arXiv [Preprint]*. arXiv: 2007.10227 doi: 10.3389/fnbot.2020.568359
- Eliasmith, C., and Anderson, C. H. (2003). *Neural engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT press.
- Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science* 338, 1202–1205. doi: 10.1126/science.1225266
- Eliasmith, C., and Trujillo, O. (2014). The use and abuse of large-scale brain models. *Curr. Opin. Neurobiol.* 25, 1–6. doi: 10.1016/j.conb.2013.09.009
- Fischl, K., Andreou, A., Stewart, T., and Fair, K. (2018). “Implementation of the neural engineering framework on the TrueNorth neurosynaptic system,” in *Proceedings of the IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Cleveland, OH. doi: 10.1109/BIOCAS.2018.8584720
- Furber, S., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638
- Gosmann, J., and Eliasmith, C. (2016). Optimizing semantic pointer representations for symbol-like processing in spiking neural networks. *PLoS One* 11:e0149928. doi: 10.1371/journal.pone.0149928.g006
- Indiveri, G., and Douglas, R. (2000). Neuromorphic vision sensors. *Science* 288, 1189–1190. doi: 10.1126/science.288.5469.1189
- 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
- Krestinskaya, O., James, A. P., and Chua, L. (2019). Neuromemristive circuits for edge computing: a review. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 4–23. doi: 10.1109/TNNLS.2019.2899262
- Krichmar, J. L., and Wagatsuma, H. (2011). *Neuromorphic and Brain-Based Robots*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511994838
- Lin, C.-K., Wild, A., Chinya, G., Cao, Y., Davies, M., Lavery, D. M., et al. (2018). Programming spiking neural networks on intel’s loihi. *Computer* 51, 52–61. doi: 10.1109/MC.2018.157113521
- Liu, S.-C., and Delbruck, T. (2010). Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* 20, 288–295. doi: 10.1016/j.conb.2010.03.007
- Liu, S.-C., Delbruck, T., Indiveri, G., Whatley, A., and Douglas, R. (2014). *Event-Based Neuromorphic Systems*. Hoboken, NJ: John Wiley & Sons. doi: 10.1002/9781118927601
- Mayr, C., Partzsch, J., Noack, M., and Schuffny, R. (2014). Configurable analog-digital conversion using the neural engineering framework. *Front. Neurosci.* 8:201. doi: 10.3389/fnins.2014.00201
- Mead, C. (1989). *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley Longman Publishing.
- Merolla, P., and Boahen, K. (2004). “A recurrent model of orientation maps with simple and complex cells,” in *Proceedings of the Advances in Neural Information Processing Systems 16*, eds S. Thrun and L. Saul (Cambridge, MA: MIT Press), 995–1002.
- Mundy, A., Knight, J. S. T., and Furber, S. (2015). “An efficient SpiNNaker implementation of the neural engineering framework,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Killarney. doi: 10.1109/IJCNN.2015.7280390
- Nagel, L., and Pederson, D. (1973). *SPICE (Simulation Program With Integrated Circuit Emphasis)* Technical Report No. UCB/ERL M382 April 1973. Berkeley, CA: University of California.
- Nichols, K., Kazmierski, T., Zwolinski, M., and Brown, A. (1994). Overview of SPICE-like circuit simulation algorithms. *IEE Proc. Circuits Devices Syst.* 141, 242–250. doi: 10.1049/ip-cds:19941246
- Stewart, T., and Eliasmith, C. (2014). Large-scale synthesis of functional spiking neural circuits. *Proc. IEEE* 102, 881–898. doi: 10.1109/JPROC.2014.2306061
- Tripathi, A., Arabizadeh, M., Khandelwal, S., and Thakur, C. S. (2019). “Analog Neuromorphic System Based on Multi Input Floating Gate MOS Neuron Model,” in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo. doi: 10.1109/ISCAS.2019.8702492
- Tsur, E. E., and Rivlin-Etzion, M. (2020). Neuromorphic implementation of motion detection using oscillation interference. *Neurocomputing* 374, 54–63. doi: 10.1016/j.neucom.2019.09.072
- van Schaik, A. (2001). Building blocks for electronic spiking neural networks. *Neural Netw.* 6, 617–628. doi: 10.1016/S0893-6080(01)00067-3
- Wang, R., Thakur, C. S., Cohen, G., Hamilton, T. J., Tapson, J., and van Schaik, A. (2017). Neuromorphic hardware architecture using the neural engineering framework for pattern recognition. *IEEE Trans. Biomed. Circuits Syst.* 11, 574–584. doi: 10.1109/TBCAS.2017.2666883
- Yang, S., Deng, B., Wang, J., Li, H., Lu, M., Che, Y., et al. (2019). Scalable digital neuromorphic architecture for large-scale biophysically meaningful neural network with multi-compartment neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 31, 148–162. doi: 10.1109/TNNLS.2019.2899936
- Yang, S., Wang, J., Deng, B., Liu, C., Li, H., Fietkiewicz, C., et al. (2018a). Real-time neuromorphic system for large-scale conductance-based spiking neural networks. *IEEE Trans. Cybern.* 49, 2490–2503. doi: 10.1109/TCYB.2018.2823730
- Yang, S., Wang, J., Lin, Q., Deng, B., Wei, X., Liu, C., et al. (2018b). Cost-efficient FPGA implementation of a biologically plausible dopamine neural network and its application. *Neurocomputing* 314, 394–408. doi: 10.1016/j.neucom.2018.07.006
- Zaidel, Y., Shalunov, A., Volinski, A., Supic, L., and Ezra Tsur, E. (2021). Neuromorphic NEF-based inverse kinematics and PID control. *Front. Neurobot.* 15:631159. doi: 10.3389/fnbot.2021.631159
- Zhang, W., Gao, B., Tang, J., Yao, P., Yu, S., Chang, M.-F., et al. (2020). Neuro-inspired computing chips. *Nat. Electron.* 3, 371–382. doi: 10.1038/s41928-020-0435-7

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 Hazan and Ezra Tsur. 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.



Real Time Generation of Three Dimensional Patterns for Multiphoton Stimulation

Paolo Pozzi^{1*} and Jonathan Mapelli^{1,2}

¹ Department of Biomedical, Metabolic and Neural Sciences, University of Modena and Reggio Emilia, Modena, Italy,

² Center for Neuroscience and Neurotechnology, University of Modena and Reggio Emilia, Modena, Italy

OPEN ACCESS

Edited by:

Michele Giugliano,
International School for Advanced
Studies (SISSA), Italy

Reviewed by:

Dimitri Tanese,
UMR8250 Neurophotonique, France
Vincent Daria,
Australian National University, Australia

*Correspondence:

Paolo Pozzi
paolo.pozzi87@unimore.it

Specialty section:

This article was submitted to
Cellular Neurophysiology,
a section of the journal
Frontiers in Cellular Neuroscience

Received: 23 September 2020

Accepted: 01 February 2021

Published: 24 February 2021

Citation:

Pozzi P and Mapelli J (2021) Real
Time Generation of Three Dimensional
Patterns for Multiphoton Stimulation.
Front. Cell. Neurosci. 15:609505.
doi: 10.3389/fncel.2021.609505

The advent of optogenetics has revolutionized experimental research in the field of Neuroscience and the possibility to selectively stimulate neurons in 3D volumes has opened new routes in the understanding of brain dynamics and functions. The combination of multiphoton excitation and optogenetic methods allows to identify and excite specific neuronal targets by means of the generation of cloud of excitation points. The most widely employed approach to produce the points cloud is through a spatial light modulation (SLM) which works with a refresh rate of tens of *Hz*. However, the computational time requested to calculate 3D patterns ranges between a few seconds and a few minutes, strongly limiting the overall performance of the system. The maximum speed of SLM can in fact be employed either with high quality patterns embedded into pre-calculated sequences or with low quality patterns for real time update. Here, we propose the implementation of a recently developed compressed sensing Gerchberg-Saxton algorithm on a consumer graphical processor unit allowing the generation of high quality patterns at video rate. This, would in turn dramatically reduce dead times in the experimental sessions, and could enable applications previously impossible, such as the control of neuronal network activity driven by the feedback from single neurons functional signals detected through calcium or voltage imaging or the real time compensation of motion artifacts.

Keywords: multiphoton microscopy, wavefront control, optogenetics, computer generated holograms, spatial light modulators, GPU (CUDA)

1. INTRODUCTION

The recent advances in the field of photonics (Pozzi et al., 2015) combined with methods of molecular (Gandolfi et al., 2017) and genetic manipulation of the samples (Boyden et al., 2005; Mutoh et al., 2012), have provided novel tools to investigate neural functions. Among these tools, optogenetics allows to selectively stimulate specific neuronal subtypes within a three-dimensional sample (Packer et al., 2013). Indeed, in order to avoid the stimulation of undesired out-of-focus cells, multiphoton stimulation is required (Papagiakoumou et al., 2010; Dal Maschio et al., 2017). The near-simultaneous stimulation of multiple cells heterogeneously distributed in three dimensions can be achieved by time multiplexing with high-speed, inertia-free scanners (Wang et al., 2011), but the only known method for truly simultaneous stimulation is the use of spatial light modulators (SLM) (Packer et al., 2012).

A coherent light source can be focused simultaneously in an arbitrary pattern of diffraction limited focal points within a three-dimensional volume through the use of a spatial light modulator in the pupil of an optical system. In order to stimulate areas wider than the diffraction limit, the technique can be combined with either temporal focusing (Pégard et al., 2017), or spiral or raster scanning (Packer et al., 2012, 2013). While this method is widely used in optogenetics, it has a variety of applications extending beyond the field of neuroscience and including optical trapping (Grier and Roichman, 2006), high throughput spectroscopy (Nikolenko et al., 2008; Gandolfi et al., 2014; Pozzi et al., 2015), and adaptive optics (Pozzi et al., 2020).

A recent publication (Zhang et al., 2018) showed how multiphoton optogenetics, applied in conjunction with multiphoton calcium imaging, can be used to manipulate in real time a network of neurons, for example clamping their calcium activity to a given threshold, or forcing cells to co-activate. However, due to the limitations in pattern calculation speeds, the method can only control the stimulation by alternating amongst a limited amount of pre-calculated patterns. True, real-time feedback-based control of a network would be greatly enhanced by the ability of calculating patterns automatically on-the-fly as they are needed.

The requirements for real-time optogenetics manipulation of calcium signals can vary widely depending on the optical setup, experiment goals, species of interest, cell type, and brain region. For the number of cells of interest and their distribution, at the state of the art for *in vivo* imaging, random access multiphoton microscopy was shown to be able to acquire signals from over five hundred cells, within an approximately 300 μm fov in all three directions at 80 Hz (Katona et al., 2012) in visual cortex. Other implementations showed performance in the same orders of magnitude, for example Bessel scanning (Lu et al., 2017) showed the activity of approximately one hundred GABA-ergic neurons at 30 Hz in the same region. In alternative samples and technologies, lightsheet microscopy in Zebrafish embryos (Wolf et al., 2015) was shown to detect signals from tens of thousands of neurons at 1 Hz from the whole embryo brain, and its acquisition frequency could increase dramatically by reducing the field of view.

As for the time resolution requirements, it mainly depends on the accuracy required for the cell response to photostimulation, as well as from the rise and fall time of calcium signals in the neurons of interest. Those in turn strongly depend on the dye or protein used for calcium imaging and on the cellular type of the neurons stimulated. Rise times are known to be generally really fast when photostimulation is activated, reaching a saturation of the signal within a couple of hundreds milliseconds. As for decay times, they are generally in the order of a second, but can go down to a few hundreds milliseconds in some transgenic mice lines (Dana et al., 2014). Even in the assumption of a calcium signal decreasing quite slowly with an exponential decay time of 1 s (corresponding to a complete return to baseline fluorescence in approximately three seconds), a signal decrease of 10% happens in the first 100 ms, which indicates the need for SLM modulation frequencies higher than 10 Hz for good optogenetic clamping of the activity. At the very limit of such scenario, cerebellar granule

cells bulk stained with Fura-2 AM dye have been shown to have, under electrical stimulation, calcium transients shorter than 200 ms from the onset to the return to baseline (Gandolfi et al., 2014), and would therefore require millisecond-scale modulation of the stimulation pattern for real-time control.

While the fields of view typical of high speed 3D calcium imaging are generally within the operating capabilities of modern SLMs, targeting hundreds of neurons with millisecond-scale modulation is a challenging endeavor. While high performance SLMs can refresh at up to hundreds of Hz, the algorithms used for computing holograms constitute the current main limitation.

For two dimensional patterns, or patterns distributed on a limited set of two-dimensional planes, relatively fast computation times can be achieved by exploiting fast Fourier transform based algorithms (Sinclair et al., 2004). However, the generation of an arbitrary 3D pattern remains the main limiting factor in the speed of operation for spatial light modulators, slowing the entire experimental procedure, and precluding any form of real-time update of three dimensional patterns. The generation of a three dimensional focusing pattern requires estimation of the phase value for each of the hundreds of thousands of pixels of the spatial light modulator maximizing the quality of the obtained pattern. The two most popular algorithms for this computation are the high-speed, lower precision random superposition (RS) algorithm, and the higher precision, lower speed Weighted Gerchberg-Saxton (WGS) algorithm (Di Leonardo et al., 2007). The RS computational cost scales linearly with $M \cdot N$, where M is the number of SLM pixels and N is the number of generated foci, while WGS scales linearly with $M \cdot N \cdot I$, where I is the number of iterations required. The quality of the hologram is generally evaluated through its efficiency (e) and uniformity (u), two metrics respectively indicating as a number between 0 and 1, the percentage of laser light actually focused in the desired locations, and the uniformity of intensities between the generated foci.

At the state of the art, when implemented with a typical SLM resolution on a consumer computer processor unit (CPU), RS can generate holograms with $e > 0.2$ and $u > 0.2$ in a few seconds, while WGS can generate holograms with $e > 0.9$ and $u > 0.9$. Unfortunately, WGS requires a few minutes for computation. Since most applications require faster computation times, it is crucial to implement such algorithms on faster time scales as it has been obtained by using a consumer graphical processors (GPU) (Bianchi and Di Leonardo, 2010). When implemented on a GPU, RS algorithm has been proved to promptly generate arbitrary patterns at video rate (Reicherter et al., 2006; Daria et al., 2009), but with its characteristic low quality. Conversely, the WGS algorithm has proven to produce high quality holograms at video rate, but only with a limited number of SLM pixels ($M < 768^2$) and on a very low number of foci ($N < 10$) (Bianchi and Di Leonardo, 2010; Vizsnyiczai et al., 2014). Additionally, although WGS results were published, no source code was openly released with them. As a result, due to the intrinsic difficulty in GPU coding, this profitable method has not yet been widely adopted, and most researchers still perform WGS computation on CPUs.

We have recently proved (Pozzi et al., 2019), how, on a CPU, a new algorithm (compressive sensing weighted Gerchberg-Saxton, CS-WGS), applying the principles of compressed sensing to the iterations of WGS can reduce its computational cost asymptotically close to the cost of RS, while maintaining the high quality of WGS holograms. Here, we present the implementation of CS-WGS on a low-cost consumer GPU, demonstrating that the algorithm is well-suited to GPU implementation, enabling video-rate computation of holograms with $e > 0.9$ and $u > 0.9$ for $N < 100$ and $M < 1,152^2$, ideally adaptable to feedback-based optogenetic control of neuronal networks.

2. METHODS

2.1. Compressive Sensing Weighted Gerchberg Saxton Algorithm

In both RS and WGS algorithms, the SLM phase pattern $\Phi^0(x', y')$ generating a set of N foci at positions $X_n = \{x_n, y_n, z_n\}$ with relative intensities $\|a_n^0\|^2$, is calculated as the phase of the interference of the N wavefronts with known phase patterns $\phi_n(x', y')$ generating each spot independently, each with a set phase delay θ_n^0 :

$$\Phi^0 = \arg \left(\sum_{n=1}^N a_n^0 e^{i(\phi_n + \theta_n^0)} \right) \quad (1)$$

where ϕ_n is defined by basic physical optics as:

$$\phi_n(x', y') = \frac{2\pi}{\lambda f} (x_n x' + y_n y') + \frac{2\pi}{\lambda f^2} (x'^2 + y'^2) z_n \quad (2)$$

In the simple random superposition algorithm, Φ^0 is simply determined through Equation (1), selecting random values for θ_n^0 . In the weighted Gerchberg-Saxton algorithm, the values of θ_n are determined through a series of alternating projections between the SLM space and the spots' positions. The algorithm begins by computation of the RS hologram Φ^0 through Equation (1). At the j -th iteration, the field E_n^j of each spot is calculated as:

$$E_n^j = \sum_{x', y' \in \Omega} A e^{-i(\Phi^{j-1} - \phi_n)} \quad (3)$$

where $\|A(x', y')\|^2$ is the distribution of light intensity at the slm surface, and Ω is the set of all SLM pixels coordinates. At this point the values of θ_n and a_n are updated as:

$$w_n^j = w_n^{j-1} \frac{\langle \|E_n^{j-1}\| \rangle_{n=1}^N}{\|E_n^{j-1}\|} \quad (4)$$

$$a_n^j = w_n^j a_n^0 \quad (5)$$

$$\theta_n^j = \arg(E_n^{j-1}) \quad (6)$$

where w_n^j are weight factors, all initialized at 1 for the first iteration. The updated values of a_n^j and θ_n^j are used to compute a new hologram Φ^j with Equation (1) and start the next iteration.

The CS-WGS algorithm is equivalent to WGS, but the summation in Equation (3) is only performed over a subset $\Omega_{compressed}^j$ of randomly distributed pixels on the SLM for $N - 2$ iterations, followed by two full iterations to ensure full convergence and the computation of phase on all SLM pixels. Conversely the value of the hologram phase can be computed, for all iterations except the last two, only for the pixels in $\Omega_{compressed}^j$. Through this adaptation, CS-WGS scales in computational cost linearly with $2 \cdot M \cdot N + c(M \cdot N \cdot (I - 2))$, where c is the ratio between the sizes of $\Omega_{compressed}^j$ and Ω .

The performance of all three described algorithms can be computed through the metrics of efficiency (e), uniformity (u), and variance (v). Efficiency is computed as the fraction of power effectively directed at the spots locations:

$$e = \sum_n I_n \quad (7)$$

where I_n is the fraction of laser intensity directed to the n -th spot. The uniformity metric is defined as:

$$u = 1 - \frac{\max_n(F_n) - \min_n(F_n)}{\max_n(F_n) + \min_n(F_n)} \quad (8)$$

where F_n is the ratio between the achieved and desired power fractions at the n -th spot:

$$F_n = \frac{I_n}{\sum_{n'} I_{n'}} / \frac{\|a_n^0\|^2}{\sum_{n'} \|a_{n'}^0\|^2} \quad (9)$$

Finally, the variance metric is expressed as the mean square relative error in the power fractions:

$$v = \frac{\sum_n (F_n - 1)^2}{N} \quad (10)$$

The efficiency metric reports on the actual fraction of power directed to the spots. It should be noted that the power fraction not directed to the spots is rarely uniformly distributed throughout the sample, and generally forms undesired excitation spots. The metric should therefore be as close to the value of 1 as possible to avoid undesired artifacts, and low values can not only be compensated by an increase in laser power.

The uniformity metric should also be as close to 1 as possible. Lower values reveal the presence of significant outliers in the spots intensities, which can lead to missing excitation of targeted cells, or to local photodamage in over-illuminated cells. Finally, the variance metric defines the general deviation of spots intensities from their desired values, and should be as close to 0 as possible in order to achieve precise control of power over all generated spots. Precise control of intensities is crucial for optogenetics stimulation, as the relative power between spots should be carefully regulated in order to prevent non-optically sectioned stimulation due to thermal effects (Picot et al., 2018).

2.2. GPU Implementation

GPU implementations of algorithms should be carefully developed in order to fully exploit the parallelized calculation performance of the devices. We report here some considerations about the implementation.

2.2.1. Global Memory Allocation

When implementing GPU code, minimization of memory transfer between the system memory and the GPU global memory is critical to achieve optimal performances. RS, WGS, and CS-WGS are all very well suited algorithms for this specific requirement, as the hologram specific inputs required are limited to the 3D coordinates of the desired spots and their desired intensities, as well as a single floating point value for the required compression factor c for CS-WGS. As most SLMs are connected to calculators as secondary monitors directly connected to the GPU, no readout of the algorithm's output to system memory is necessary, but the hologram is directly projected on the SLM through CUDA-OPENGL interoperability.

Additionally, some fixed parameters characterizing the physical and geometrical properties of the SLM and the optical system (e.g., the coordinates x', y' of the SLM pixels, the phase to gray scale lookup table of the SLM output), are uploaded to the GPU only once at startup and used for all holograms computed during an experimental session. Such initialization does not therefore affect the speed of the algorithm convergence.

2.2.2. Backwards Propagation of RS and WGS

Given, for each spot, the values of the desired coordinates and intensities X_n, a_n^0 , weights w_n^j and phase terms θ_n^j , at each iteration the hologram phase is computed according to Equation (1). Each of the parallel threads of the GPU evaluates the equation for one of the M pixels of the SLM, performing the summation over all spots. Counter-intuitively, the values of ϕ_n are computed at each iteration according to Equation (2), instead of computed once and stored in global memory, as their direct computation is significantly faster than accessing values stored in the GPU global memory.

The obtained hologram Φ^j is stored in a pre-allocated section of global memory, or, in case of the last iteration, copied to an OpenGL texture buffer, and projected on the SLM surface. It should be noticed that vertical synchronization in the OpenGL environment should be enabled, in order to avoid artifacts during the alternation of different holograms on the SLM. As a consequence, the total time required for the last iteration will be extended until the next refresh of the SLM screen.

2.2.3. Forward Propagation of RS and WGS

Given an hologram Φ^j , and the known intensity distribution of light at the SLM surface, the field at each spot can be computed through Equation (3), which therefore requires the sum of M complex numbers per each spot. This sort of computation is known in GPU programming as a dimensionality reduction, and is performed by using k threads to iteratively perform the sum of M/k elements of the sum, until the amount of elements to be summed equals one. Since a modern GPU can run 1,024 threads in one block, and the number of SLM pixels in the

system aperture is $<1,024^2$, the dimensionality reduction always converged in two iterations for the presented results.

2.2.4. Compressed Sensing

During initialization, all arrays containing data referring to SLM pixels (e.g., hologram phase, known intensity at the pupil) are reorganized in a randomly selected order. At each iteration only $c \cdot M$ GPU threads are employed both for forwards and backwards projection, performing computation on pixels which will be adjacent in GPU global memory for optimal performance, but randomly distributed in the pupil due to the random reorganization. Only the backwards projection at the very last iteration is performed on all pixels, in order to compute the phase of the full hologram. The actual position in the pupil for each pixel is stored during initialization in an additional array in global memory, and used at the end of the computation to apply the correct phase values to the correct OpenGL texture pixels for projection.

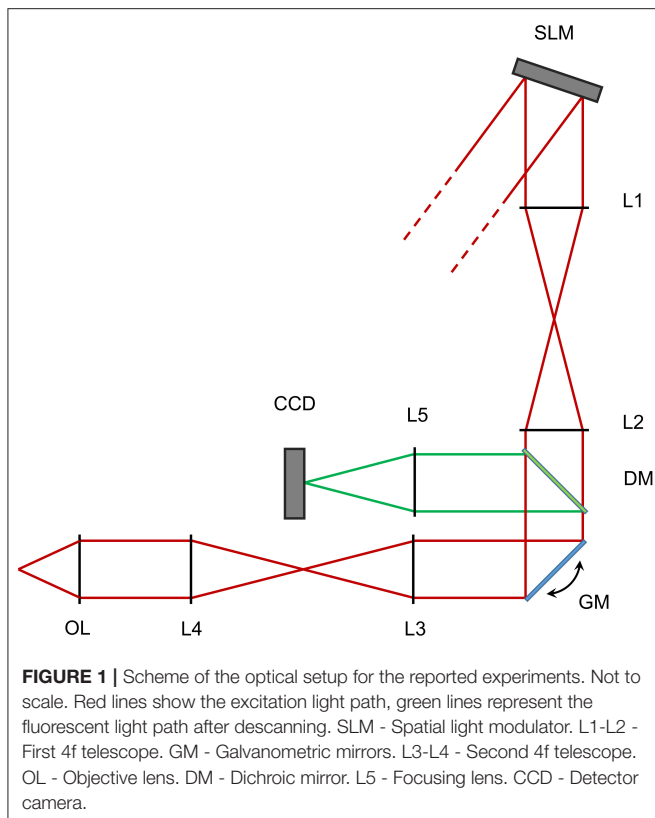
2.3. Experimental Setup

Holograms were computed on a budget desktop GPU (GTX1050, Nvidia), also available in several mid-range laptops. Experimental results were obtained by measuring two-photon excited fluorescence from a solid, 1.7 mm thick fluorescent slide (FSK-2, Thorlabs, USA) on a custom system for multiphoton imaging and optogenetics. The system includes an SLM with a refresh frequency of 31 Hz, and a panel of $1,152 \times 1,920$ pixels, with pixel pitch of $9.2\mu\text{m}$ (Meadowlark, USA), with the short side optically matched to the round aperture of the optical system, limiting hologram computation to a round sub-region of 1,152 pixels in diameter.

The source employed is a Ti:Sa laser (Chameleon Ultra II, Coherent, USA), tuned to 800 nm, expanded through a telescope of two infrared achromatic doublets (AC-127-050-B and AC-254-250-B, Thorlabs) to a beam waist radius of 6 mm at the SLM panel. A simplified scheme of the setup is shown in **Figure 1**.

The spatial light modulator (SLM) surface is conjugated to a couple of silver coated galvanometric mirrors (GM, GVS-012/M, Thorlabs, USA) by a 4-f beam reducing telescope of two infrared achromatic doublets (L1 and L2, AC-508-200-B and AC-508-150-B, Thorlabs). A custom made glass slide with a 0.5 mm round deposition of titanium is placed in the focal plane of the first lens in order to block the 0-th order of diffraction of the SLM while minimally affecting the projected pattern. We were in fact unable to measure any differences in spots intensities when adding and removing the blocker. The Galvanometric mirrors are conjugated through a beam expanding 4-f telescope of broad spectrum achromatic doublets (L3 and L4, AC-508-180-AB and AC-508-400-AB) to the back aperture of a water dipping microscope objective (OL, XLUMPlanFL N, 20X, 1.0 NA, Olympus, Japan). In this configuration, a phase-conjugated image of the SLM is produced on the back aperture of the objective with a magnification of 5:3, so that the 10.6 mm side of the SLM is matched with the 18 mm aperture of the objective.

Fluorescence light is reflected by a longpass dichroic mirror (DM, FF665-Di02-25x36, Semrock, USA) and further filtered from laser light through an IR-blocking filter (FF01-680/SP-25,



Semrock, USA). The mirrors are conjugated by a couple of 4-f telescopes of visible achromatic doublets and a custom channel splitter (not shown) with a mounted 12 – 72 mm, 1.2f# zoom lens (L9, Cosina, Sony, Japan), mounted on a high speed, 128 × 128 pixels EMCCD camera (CCD, Hnu 128 AO, Nuvu, Canada).

The focal and aperture of the camera zoom lens are chosen in order to image a field of view of $400\mu\text{m} \times 400\mu\text{m}$ for two color channels in 64×64 pixels subregions of the camera sensor, while maintaining a depth of field of $400\mu\text{m}$ in order to visualize three-dimensional patterns without defocus aberrations. Focusing of the laser in the fluorescent slide generates two-photon fluorescence, the intensity of which increases quadratically with local power, and is therefore an appropriate reporter of the stimulation intensity which could be achieved in a biological sample.

Measurements were performed at approximately $300\mu\text{m}$ depth within the fluorescent slide, in order to avoid spots generated at high axial distances from the focal plane to be focused outside the sample. The galvanometric mirrors were operated in a $50\mu\text{m}$ wide constant speed spiral scan at 120 Hz throughout the experiments, in order to minimize photobleaching effects, as well as compensating for local inhomogeneities of the fluorescent slide. The descanned nature of the detection light path insured that the motion of the mirrors did not affect the shape of the spots at the detector.

3. RESULTS

In order to compute convergence timing for RS, WGS, and CS-WGS algorithms, two types of holograms were computed: regular two-dimensional grids of uniform spots, considered as a worst case scenario for pattern uniformity, and a more realistic random distributions of spots of varying intensity within a cubic volume of $200\mu\text{m}$. Grids were calculated for square patterns from 4 to 144 spots. Random distributions were calculated from 9 to 99 spots. Lower amounts of spots were not considered, as SLMs have generally unreliable performance independently from the algorithm used when generating very few spots. If possible, in such situation, other excitation methods should be preferred (e.g., acousto-optic scanners). A maximum performance reference was computed through 200 iterations of WGS. Holograms for the same distributions of points were then calculated with RS, with WGS, and with CS-WGS for compression factors ranging from 2^{-1} to 2^{-8} . WGS and CS-WGS computations were repeated for an increasing number of iterations, until a uniformity value higher than a target percentage of the maximum performance was reached. **Figure 2** shows the timings required for full convergence of the algorithms, as well as a comparison between the uniformity performances achieved by the non-iterative RS compared to the iterative algorithms. Only the best performing value of the compression factor in CS-WGS is reported for each data point. For these results, vertical synchronization of the GPU with the SLM screen was disabled, in order to present data unaffected by the specific hardware employed. The data reported clearly shows how, in any of the presented scenarios, CS-WGS greatly outperforms WGS, with generally half the convergence time, and up to a factor 5 speedup when computing holograms for regular lattices of high numbers of spots. This, while being an unlikely pattern for optogenetics experiments, is often required for imaging or optical trapping applications.

While still significant, the lowest performance advantage of CS-WGS over WGS, was observed for random distributions of small numbers of spots (<50) for relatively low performance targets (<92% of full convergence uniformity) for which WGS converged in only two iterations, leaving small space for improvement with the application of compressed sensing. In this situation, WGS still resulted 1.5 times slower than CS-WGS.

It should be noticed how, while a GPU implementation of RS remains up to an order of magnitude faster than iterative algorithms, the uniformity of the patterns produced can be extremely low for any number of spots, and this algorithm should only be used when the experimental scenario requires extremely high computation speed for a very high number of spots.

A more realistic utilization scenario for high speed hologram computation, however, is one in which the full convergence performance is sacrificed in order to achieve computation times equivalent to the refresh rate of the SLM, in order to update the hologram on-the-fly as fast as the hardware allows it. Fixed refresh rate performance of RS, WGS, and CS-WGS algorithms was measured both through calculation of the theoretical efficiency and uniformity of the patterns, and by visualization of multiphoton fluorescence excitation in the experimental setup. In these measurements, vertical synchronization of the GPU

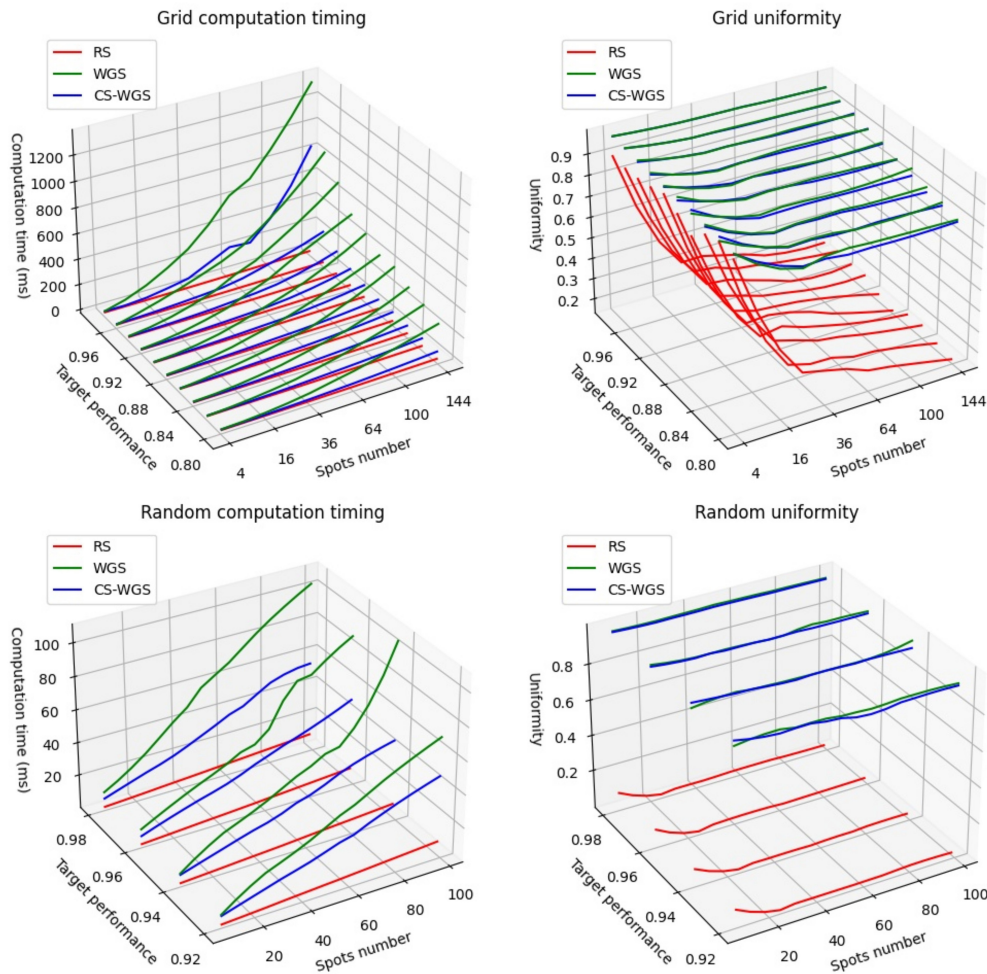


FIGURE 2 | Performance comparison of the algorithms when computing patterns at a set fraction of the full convergence uniformity.

with the SLM screen was enabled, as it is required for correct experimental application. The SLM used for data validation was capable of a refresh rate of 31 Hz. However, hologram computing times were constrained to a refresh rate of 15 Hz, as it was experimentally found that, while operating at the SLM limit of 31 Hz, the quality of the projected pattern was strongly dependent on the pixel response times of the SLM at the experimental wavelength, and comparison of experimental data resulted difficult. The performance of CS-WGS was computationally tested for a range of compression rates c from 2^{-1} to 2^{-8} . The best performing compression rate for the uniformity metric was used for experimental comparison. An additional set of measurements for full convergence of WGS was added in order to provide a reference for the best achievable pattern quality without frame rate constraints.

Tests were performed in three critical scenarios for multi-foci real-time computation. The first two were two-dimensional, regularly spaced, grids of points rotating in 3D space, representing a worst-case scenario for pattern uniformity. The two grids differ in number of total spots, one is a grid of 100 spots,

for which WGS could only perform a single iteration within the 64 ms frame time limit, the other is a more limited 36 spots grid, for which WGS could achieve 5 full iterations. The third scenario was a more realistic distribution of 100 points in a random pattern, within a cubic volume of side $300\mu\text{m}$, with randomly distributed target intensities.

The computed efficiencies and intensities achievable with a 15 Hz frame rate are reported in **Figure 3**. Error bars were calculated from the standard deviation of the mean performance over 10 calculations with different initial values of θ_n^0 and different spatial orientations of the patterns. It can be observed how, for a large amount of regularly spaced spots, WGS has practically no advantage over RS, due to the limited amount of iterations which can be performed within the time limit.

The performance of WGS improve for smaller amounts of spots and less regular patterns, but CS-WGS still stands out as the better performing algorithm in all scenarios. Low compression rates of CS-WGS tend to prioritize uniformity, due to their better sampling of the pupil, while high compression rates tend to prioritize efficiency due to the higher number of iterations

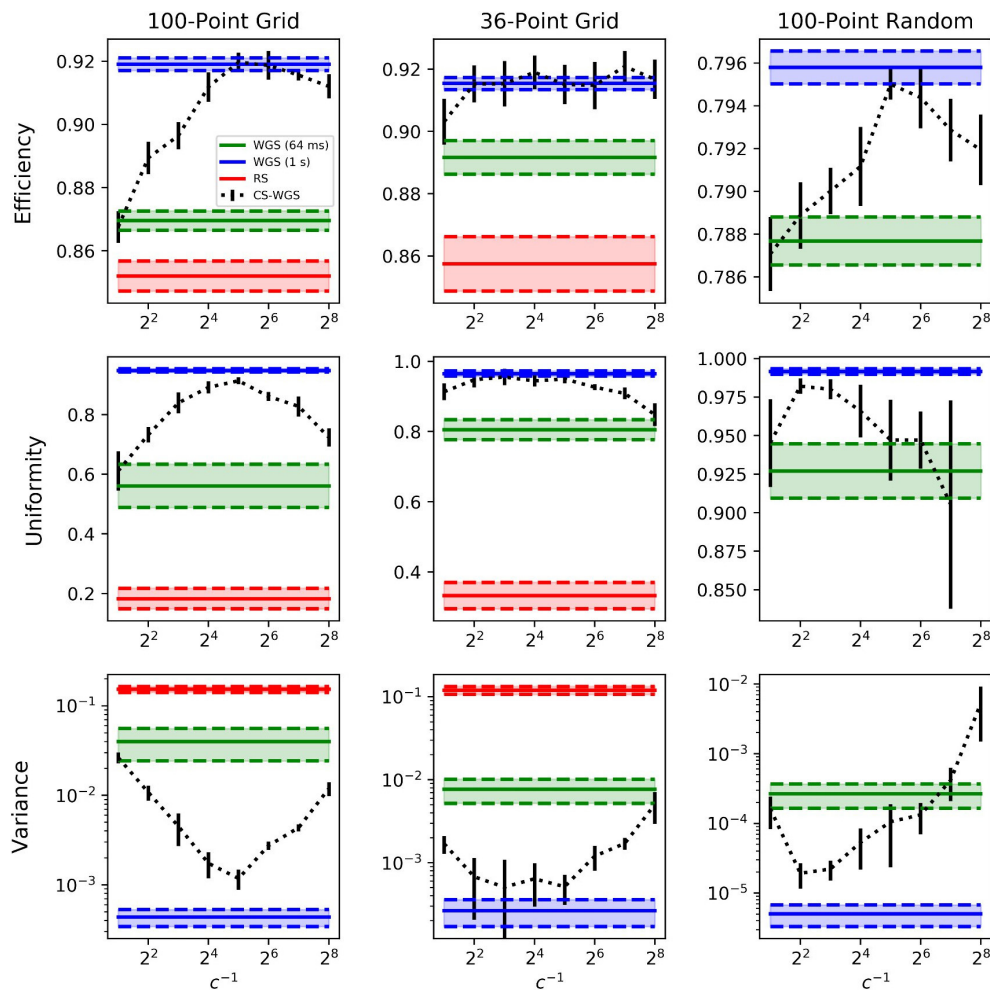


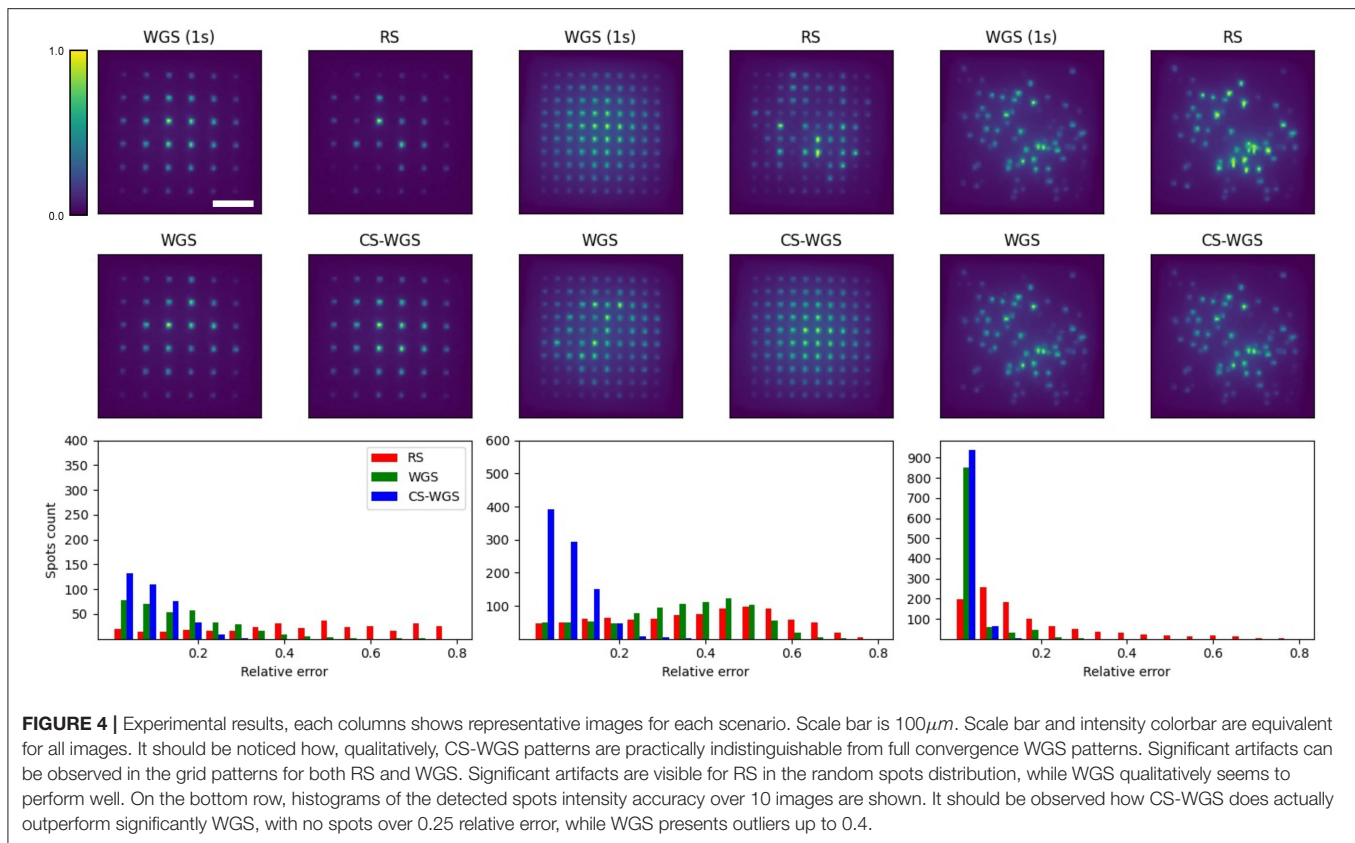
FIGURE 3 | Performance comparison of the algorithms when computing in real time at 15Hz in selected scenarios. The legend is valid for all graphs. RS results for the 100 points random distribution has been omitted as out of a reasonable graph scale at $e = 0.554 \pm 0.004$, $u = 0.0245 \pm 0.004$ and $v = 0.44 \pm 0.22$. Similarly, the CS-WGS uniformity for the same pattern at 2^{-8} compression was omitted, scoring $u = 0.46 \pm 0.17$. RS computation times were approximately 25ms for 100 spots patterns, and 12ms for 36 spots patterns. Error bars report standard deviation.

achievable. Nonetheless, unless extreme compression factors were used for spots patterns with varying intensities, CS-WGS provides better performance than WGS in all tested scenarios. Results equal or similar to a fully converging implementation of WGS could be achieved in all tested scenarios for well-tuned compression factors.

Since experimental systems are non-ideal, often the performance of the computed patterns can be affected by the experimental setup (Palima and Daria, 2006). In order to prove the improvement in performance provided by CS-WGS is detectable and significant in experimental scenarios, we provided verification of the results of **Figure 3** on the setup described in the methods section.

Experimental results are reported in **Figure 4**. All holograms show a decrease in signal intensity toward the edges of the frame, due to the loss in diffraction efficiency of the SLM at the edges of its addressable volume, which is independent

from the algorithm's performance. Images are reported with a 10X upscaling with bilinear filtering in order to reduce aliased sampling artifacts due to the sensor's low resolution. For each experimental scenario, 10 different variants of the pattern were computed by rotating the grids in three dimensions and rearranging the spots random distribution. In order to estimate the intensity of the spots, a blob detection algorithm was run over images acquired from the camera, integrating the pixels intensities within the blob. blob locations and sizes were estimated over the average of 10 images of the WGS pattern with full convergence and used to compute intensities for the other algorithms. The relative error of each spot's intensity was computed from the ratio of its blob intensity compared to that of the fully converging WGS pattern. It should be noticed the intensity detection error on a fixed pattern could get up to 5% root mean square, depending on the intensity of the spot. As expected, RS performed the worst, with average errors of $0.47 \pm$



0.30 for the 36 spots grid, 0.32 ± 0.11 for the 100 spots grids, and 0.17 ± 0.02 for the random patterns. When constrained to 64 ms of computation time, WGS performed similarly to RS when computing the 100 spots grid, with an average error of 0.28 ± 0.03 , due to its inability to perform more than two iterations in the given time. It performed better for the 100 spots random pattern and the 36 spots grid (respectively 0.03 ± 0.02 and 0.16 ± 0.03 relative errors). Still, CS-WGS proved to provide the best performance in all scenarios, with average errors of 0.08 ± 0.01 for the 36 spots grid, 0.06 ± 0.03 for the 100 spots grids, and 0.02 ± 0.01 for the random patterns. More importantly the highest outliers for all patterns for RS reached relative errors of 0.8, meaning the spot was either almost completely missing or nearly twice as bright as it should have been. Outliers for WGS reached up to 0.8 for the 100 points grid, up to 0.6 for the 36 spots grid and up to 0.4 for the random distribution. Conversely, in all scenarios CS-WGS managed to keep all spots under 0.25 relative error. Computing the same pattern multiple times with different initialization phases led to similar statistics in error distributions. Most importantly, the outlier spots would be positioned in random, unpredictable positions within the pattern.

It should be noted how for the worst case scenario of regular grid patterns, significant deviations from the desired patterns can easily be noticed in the intensity distributions of RS and WGS, while CS-WGS seems indistinguishable from the desired pattern, as highlighted by the numerical metrics. In the random distribution pattern, RS is still visibly inaccurate, while WGS and

CS-WGS seem to perform equivalently. However, the numerical metrics highlight how CS-WGS holograms present smaller deviations from the desired pattern, and therefore provide the best achievable performance within the time constraint.

Examples of real time manipulation of the patterns are available as **Supplementary Materials**, showing the selected patterns rotating in three dimensions through real-time recalculation. The videos show how smooth live update of the hologram is possible, with reasonably constant performance throughout the experiment.

From the results, it is apparent that the compression factor and number of iterations can be fine-tuned to achieve maximum performance. However, this is often not possible for real time generation of generic patterns with varying numbers of spots or geometrical distribution. In such a situation, a compression factor between 1/8 and 1/16 seems to provide a good baseline value to achieve reliable performance in a variety of experimental conditions.

4. DISCUSSION

In this manuscript a GPU implementation of the CS-WGS algorithm is presented, and benchmarked against the two most popular alternatives available, being RS and WGS. The results clearly show how the higher convergence speed of CS-WGS, makes it the ideal candidate for real-time applications. The GPU implementation of the algorithm proves, for real time

applications, absolutely necessary, as similar spots patterns to those tested would require several seconds for computation with CS-WGS (Pozzi et al., 2019), and up to several minutes with WGS.

While the presented experimental tests were limited by the refresh rate of the available SLM, the algorithm could easily be used to control even faster systems, provided a reasonable amount of spots is selected, and the compression factor is tuned accordingly. The ability of computing high quality holograms in real time could enable real-time, feedback-based control of neuronal networks, driven by calcium (Lu et al., 2017) or voltage activity (Gandolfi et al., 2015) without being limited to stimulation on pre-calculated spatial patterns.

As an example of the advantages of real-time computation compared to the use of pre-computed patterns in closed loop stimulation, keeping N cells clamped at the same level of activity through pre-computed patterns by binary switching of photostimulation on each cell, would require pre-calculation of patterns stimulating all possible combinations of at least one of the N cells. In practice, this means that $2^N - 1$ patterns would be required, limiting the applicability of the experiment to only a very few neurons.

A similar consideration can be made for the possibility of synchronizing the activity of cell populations to a single “trigger neuron.” For N selected trigger neurons, at least 2^N patterns would need to be calculated, or more if any neuron would need to be coupled with two separate trigger neurons.

It should be acknowledged that fast photoswitching of single points in a given fixed pattern can be achieved by the use of a digital micromirror device in the image plane (Go et al., 2013) to modulate intensity. However, this is still limited in the number of available patterns in the DMD memory (a few tens to a few hundreds, depending on the hardware used), has limited axial positioning extent (only $\pm 10\mu\text{m}$ in the reported publication), it would not work for spots located at similar lateral positions but at different axial depths, and in general requires significant modifications to a standard SLM based setup, when compared to a simpler modification of software. Moreover, due to the accuracy of our algorithm in the modulation of power of single spots, even stimulation based on analog modulation of the excitation power for each spot, instead of a binary on/off behavior, could be implemented.

Independently from closed loop photostimulation, an immediate outcome of this implementation lies in the extreme streamlining of the experimental procedure, practically eliminating any waiting time between the selection of the point of interests and the experimental procedure. Of note, it can be extremely useful for *in-vivo* recordings with awake mice. In these circumstances, experiments are in fact extremely time-sensitive, and the minimization of the experiments duration is of utmost importance.

REFERENCES

- Bianchi, S., and Di Leonardo, R. (2010). Real-time optical micro-manipulation using optimized holograms generated on the GPU. *Comput. Phys. Commun.* 181, 1444–1448. doi: 10.1016/j.cpc.2010.04.012
- Boyden, E. S., Zhang, F., Bamberg, E., Nagel, G., and Deisseroth, K. (2005). Millisecond-timescale, genetically targeted optical control of neural activity. *Nat. Neurosci.* 8, 1263–1268. doi: 10.1038/nn1525
- Dal Maschio, M., Donovan, J. C., Helmbrecht, T. O., and Baier, H. (2017). Linking neurons to network function and behavior by two-photon
- Furthermore, the newly introduced ability of updating the pattern in real-time at the SLM refresh speed limit can potentially enable previously impossible experimental protocols. For instance, the correction of motion artifacts, which is currently performed only through the use of scanners and focus actuators, for rigid linear movements (Vladymyrov et al., 2016), could be enabled for sample rotations and non rigid deformations through SLM patterns adaptation.
- Since GPU programming is not a widespread practice amongst the optics and neuroscience research community, the software used to generate the results presented in the paper is made available as a free and open-source library (Pozzi, 2020) for non commercial purposes, to ensure a widespread adoption of the method. The software library is compatible with all SLMs controlled as external screen, and is not necessarily limited to 64 ms computation time. Some modifications to the code may be required to directly drive SLMs with dedicated pci-e interfaces. The software consists in Python (Van Rossum and Drake, 1995) code controlling the GPU using CUDA (Nickolls et al., 2008) through the PyCuda (Klöckner et al., 2012) library and rendering holograms directly to the SLM through the GLFW OpenGL framework.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/Supplementary Material.

AUTHOR CONTRIBUTIONS

PP designed the research, performed the experiments, and wrote the first version of the manuscript. JM designed the research and contributed to the manuscript writing. All authors contributed to the article and approved the submitted version.

FUNDING

The work is partially funded by the SMART BRAIN project. SMART-BRAIN is a Partnering Project to the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fncel.2021.609505/full#supplementary-material>

- holographic optogenetics and volumetric imaging. *Neuron* 94, 774–789. doi: 10.1016/j.neuron.2017.04.034
- Dana, H., Chen, T.-W., Hu, A., Shields, B. C., Guo, C., Looger, L. L., et al. (2014). Thyl-gcamp6 transgenic mice for neuronal population imaging *in vivo*. *PLoS ONE* 9:e108697. doi: 10.1371/journal.pone.0108697
- Daria, V. R., Stricker, C., Bowman, R., Redman, S., and Bachor, H.-A. (2009). Arbitrary multisite two-photon excitation in four dimensions. *Appl. Phys. Lett.* 95:093701. doi: 10.1063/1.3216581
- Di Leonardo, R., Ianni, F., and Ruocco, G. (2007). Computer generation of optimal holograms for optical trap arrays. *Opt. Exp.* 15, 1913–1922. doi: 10.1364/OE.15.001913
- Gandolfi, D., Cerri, S., Mapelli, J., Polimeni, M., Tritto, S., Fuzzati-Armentero, M.-T., et al. (2017). Activation of the CREB/c-Fos pathway during long-term synaptic plasticity in the cerebellar granular layer. *Front. Cell. Neurosci.* 11:184. doi: 10.3389/fncel.2017.00184
- Gandolfi, D., Mapelli, J., and D'Angelo, E. (2015). Long-term spatiotemporal reconfiguration of neuronal activity revealed by voltage-sensitive dye imaging in the cerebellar granular layer. *Neural Plast.* 2015:284986. doi: 10.1155/2015/284986
- Gandolfi, D., Pozzi, P., Tognolina, M., Chirico, G., Mapelli, J., and D'Angelo, E. (2014). The spatiotemporal organization of cerebellar network activity resolved by two-photon imaging of multiple single neurons. *Front. Cell. Neurosci.* 8:92. doi: 10.3389/fncel.2014.00092
- Go, M. A., To, M.-S., Stricker, C., Redman, S., Bachor, H.-A., Stuart, G., et al. (2013). Four-dimensional multi-site photolysis of caged neurotransmitters. *Front. Cell. Neurosci.* 7:231. doi: 10.3389/fncel.2013.00231
- Grier, D. G., and Roichman, Y. (2006). Holographic optical trapping. *Appl. Opt.* 45, 880–887. doi: 10.1364/AO.45.000880
- Katona, G., Szalay, G., Maák, P., Kaszás, A., Veress, M., Hillier, D., et al. (2012). Fast two-photon *in vivo* imaging with three-dimensional random-access scanning in large tissue volumes. *Nat. Methods* 9, 201–208. doi: 10.1038/nmeth.1851
- Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., and Fasih, A. (2012). Pycuda and pyopencl: a scripting-based approach to GPU run-time code generation. *Parall. Comput.* 38, 157–174. doi: 10.1016/j.parco.2011.09.001
- Lu, R., Sun, W., Liang, Y., Kerlin, A., Bierfeld, J., Seelig, J. D., et al. (2017). Video-rate volumetric functional imaging of the brain at synaptic resolution. *Nat. Neurosci.* 20, 620–628. doi: 10.1038/nn.4516
- Mutoh, H., Akemann, W., and Knopfel, T. (2012). Genetically engineered fluorescent voltage reporters. *ACS Chem. Neurosci.* 3, 585–592. doi: 10.1021/cn300041b
- Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue* 6, 40–53. doi: 10.1145/1365490.1365500
- Nikolenko, V., Watson, B. O., Araya, R., Woodruff, A., Peterka, D. S., and Yuste, R. (2008). Slm microscopy: scanless two-photon imaging and photostimulation using spatial light modulators. *Front. Neural Circuits* 2:5. doi: 10.3389/neuro.04.005.2008
- Packer, A. M., Peterka, D. S., Hirtz, J. J., Prakash, R., Deisseroth, K., and Yuste, R. (2012). Two-photon optogenetics of dendritic spines and neural circuits. *Nat. Methods* 9:1202. doi: 10.1038/nmeth.2249
- Packer, A. M., Roska, B., and Häusser, M. (2013). Targeting neurons and photons for optogenetics. *Nat. Neurosci.* 16:805. doi: 10.1038/nn.3427
- Palima, D., and Daria, V. R. (2006). Effect of spurious diffraction orders in arbitrary multifoci patterns produced via phase-only holograms. *Appl. Opt.* 45, 6689–6693. doi: 10.1364/AO.45.006689
- Papagiakoumou, E., Anselmi, F., Bégue, A., De Sars, V., Glückstad, J., Isacoff, E. Y., et al. (2010). Scanless two-photon excitation of channelrhodopsin-2. *Nat. Methods* 7, 848–854. doi: 10.1038/nmeth.1505
- Pégard, N. C., Mardinly, A. R., Oldenburg, I. A., Sridharan, S., Waller, L., and Adesnik, H. (2017). Three-dimensional scanless holographic optogenetics with temporal focusing (3D-shot). *Nat. Commun.* 8, 1–14. doi: 10.1038/s41467-017-01031-3
- Picot, A., Dominguez, S., Liu, C., Chen, I.-W., Tanese, D., Ronzitti, E., et al. (2018). Temperature rise under two-photon optogenetic brain stimulation. *Cell Rep.* 24, 1243–1253. doi: 10.1016/j.celrep.2018.06.119
- Pozzi, P. (2020). *SLM-3dPointCloud*. Available online at: <https://github.com/ppozzi/SLM-3dPointCloud>
- Pozzi, P., Gandolfi, D., Tognolina, M., Chirico, G., Mapelli, J., and D'Angelo, E. (2015). High-throughput spatial light modulation two-photon microscopy for fast functional imaging. *Neurophotonics* 2:015005. doi: 10.1117/1.NPh.2.1.015005
- Pozzi, P., Maddalena, L., Ceffa, N., Soloviev, O., Vdovin, G., Carroll, E., et al. (2019). Fast calculation of computer generated holograms for 3D photostimulation through compressive-sensing Gerchberg-Saxton algorithm. *Methods Protoc.* 2:2. doi: 10.3390/mps2010002
- Pozzi, P., Smith, C., Carroll, E., Wilding, D., Soloviev, O., Booth, M., et al. (2020). Anisoplanatic adaptive optics in parallelized laser scanning microscopy. *Opt. Exp.* 28, 14222–14236. doi: 10.1364/OE.389974
- Reicherter, M., Zwick, S., Haist, T., Kohler, C., Tiziani, H., and Osten, W. (2006). Fast digital hologram generation and adaptive force measurement in liquid-crystal-display-based holographic tweezers. *Appl. Opt.* 45, 888–896. doi: 10.1364/AO.45.000888
- Sinclair, G., Leach, J., Jordan, P., Gibson, G., Yao, E., Laczik, Z. J., et al. (2004). Interactive application in holographic optical tweezers of a multi-plane Gerchberg-Saxton algorithm for three-dimensional light shaping. *Opt. Exp.* 12, 1665–1670. doi: 10.1364/OPEX.12.001665
- Van Rossum, G., and Drake F. L. Jr. (1995). *Python Tutorial*. Amsterdam: Centrum voor Wiskunde en Informatica.
- Vizsniczai, G., Kelemen, L., and Ormos, P. (2014). Holographic multi-focus 3d two-photon polymerization with real-time calculated holograms. *Opt. Exp.* 22, 24217–24223. doi: 10.1364/OE.22.024217
- Vladymyrov, M., Abe, J., Moalli, F., Stein, J. V., and Ariga, A. (2016). Real-time tissue offset correction system for intravital multiphoton microscopy. *J. Immunol. Methods* 438, 35–41. doi: 10.1016/j.jim.2016.08.004
- Wang, K., Liu, Y., Li, Y., Guo, Y., Song, P., Zhang, X., et al. (2011). Precise spatiotemporal control of optogenetic activation using an acousto-optic device. *PLoS ONE* 6:e28468. doi: 10.1371/journal.pone.0028468
- Wolf, S., Supatto, W., Debrégeas, G., Mahou, P., Kruglik, S. G., Sintes, J.-M., et al. (2015). Whole-brain functional imaging with two-photon light-sheet microscopy. *Nat. Methods* 12, 379–380. doi: 10.1038/nmeth.3371
- Zhang, Z., Russell, L. E., Packer, A. M., Gauld, O. M., and Häusser, M. (2018). Closed-loop all-optical interrogation of neural circuits *in vivo*. *Nat. Methods* 15, 1037–1040. doi: 10.1038/s41592-018-0183-z

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 Pozzi and Mapelli. 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.



Hardware Design for Autonomous Bayesian Networks

Rafatul Faria^{1*}, Jan Kaiser¹, Kerem Y. Camsari² and Supriyo Datta¹

¹ Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States, ² Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States

Directed acyclic graphs or Bayesian networks that are popular in many AI-related sectors for probabilistic inference and causal reasoning can be mapped to probabilistic circuits built out of probabilistic bits (p-bits), analogous to binary stochastic neurons of stochastic artificial neural networks. In order to satisfy standard statistical results, individual p-bits not only need to be updated sequentially but also in order from the parent to the child nodes, necessitating the use of sequencers in software implementations. In this article, we first use SPICE simulations to show that an autonomous hardware Bayesian network can operate correctly without any clocks or sequencers, but only if the individual p-bits are appropriately designed. We then present a simple behavioral model of the autonomous hardware illustrating the essential characteristics needed for correct sequencer-free operation. This model is also benchmarked against SPICE simulations and can be used to simulate large-scale networks. Our results could be useful in the design of hardware accelerators that use energy-efficient building blocks suited for low-level implementations of Bayesian networks. The autonomous massively parallel operation of our proposed stochastic hardware has biological relevance since neural dynamics in brain is also stochastic and autonomous by nature.

Keywords: Bayesian network, probabilistic spin logic, binary stochastic neuron, magnetic tunnel junction, inference

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Francesco Maria Puglisi,
University of Modena and Reggio
Emilia, Italy
Alexantrou Serb,
University of Southampton,
United Kingdom

*Correspondence:

Rafatul Faria
rafatul.faria@gmail.com

Received: 18 July 2020

Accepted: 26 January 2021

Published: 08 March 2021

Citation:

Faria R, Kaiser J, Camsari KY and
Datta S (2021) Hardware Design for
Autonomous Bayesian Networks.
Front. Comput. Neurosci. 15:584797.
doi: 10.3389/fncom.2021.584797

1. INTRODUCTION

Bayesian networks (BN) or belief nets are probabilistic directed acyclic graphs (DAG) popular for reasoning under uncertainty and probabilistic inference in real-world applications such as medical diagnosis (Nikovski, 2000), genomic data analysis (Friedman et al., 2000; Jansen et al., 2003; Zou and Conzen, 2004), forecasting (Sun et al., 2006; Ticknor, 2013), robotics (Premebida et al., 2017), image classification (Arias et al., 2016; Park, 2016), neuroscience (Bielza and Larrañaga, 2014), and so on. BNs are composed of probabilistic nodes and edges from *parent* to *child* nodes and are defined in terms of conditional probability tables (CPT) that describe how each *child node* is influenced by its *parent nodes* (Heckerman and Breese, 1996; Koller and Friedman, 2009; Pearl, 2014; Russell and Norvig, 2016). The CPTs can be obtained from expert knowledge and/or machine learned from data (Darwiche, 2009). Each node and edge in a BN have meaning representing specific probabilistic events and their conditional dependencies and they are easier to interpret (Correa et al., 2009) than neural networks where the hidden nodes do not necessarily have meaning. Unlike neural networks where useful information is extracted only at the output nodes for prediction purposes, BNs are useful for both prediction and inference by looking at not only the output nodes but also other nodes of interest. Computation of different probabilities from a

BN becomes intractable when the network gets deeper and more complicated with child nodes having many parent nodes. This has inspired various hardware implementations of BNs for efficient inference (Rish et al., 2005; Chakrapani et al., 2007; Weijia et al., 2007; Jonas, 2014; Querlioz et al., 2015; Zermani et al., 2015; Behin-Aein et al., 2016; Friedman et al., 2016; Thakur et al., 2016; Tylman et al., 2016; Shim et al., 2017). In this article, we have elucidated the design criteria for an autonomous (clockless) hardware for BN unlike other implementations that typically use clocks.

Recently, a new type of hardware computing framework called probabilistic spin logic (PSL) is proposed (Camsari et al., 2017a) based on a building block called probabilistic bits (p-bits) that are analogous to binary stochastic neurons (BSN) (Ackley et al., 1985; Neal, 1992) of the artificial neural network (ANN) literature. p-bits can be interconnected to solve a wide variety of problems such as optimization (Sutton et al., 2017; Borders et al., 2019), inference (Faria et al., 2018), an enhanced type of Boolean logic that is invertible (Camsari et al., 2017a; Faria et al., 2017; Pervaiz et al., 2017, 2018), quantum emulation (Camsari et al., 2019), and *in situ* learning from probability distributions (Kaiser et al., 2020).

Unlike conventional deterministic networks built out of deterministic, stable bits, stochastic or probabilistic networks composed of p-bits (**Figure 1A**), can be correlated by interconnecting them to construct p-circuits defined by two equations (Ackley et al., 1985; Neal, 1992; Camsari et al., 2017a): (1) a p-bit/BSN equation and (2) a weight logic/synapse equation. The output of a p-bit, m_i , is related to its dimensionless input I_i by the equation:

$$m_i(t + \tau_N) = \text{sgn}(\text{rand}(-1, 1) + \tanh I_i(t)) \quad (1a)$$

where $\text{rand}(-1, +1)$ is a random number uniformly distributed between -1 and $+1$, and τ_N is the neuron evaluation time.

The synapse generates the input I_i from a weighted sum of the states of other p-bits. In general, the synapse can be a linear or non-linear function, although a common form is the linear synapse described according to the equation:

$$I_i(t + \tau_S) = I_0 \left(h_i + \sum_j J_{ij} m_j(t) \right) \quad (1b)$$

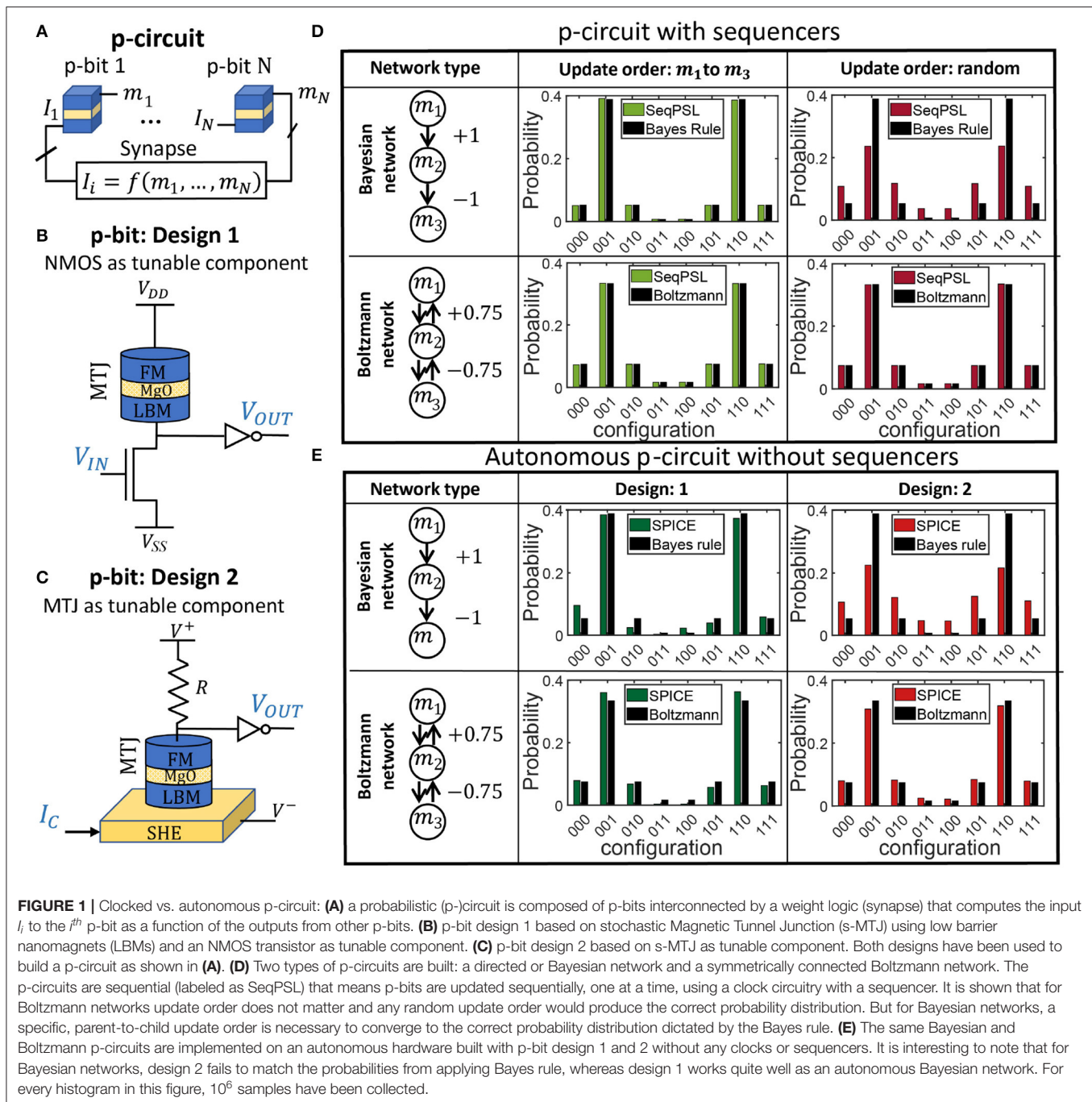
where h_i is the on-site bias and J_{ij} is the weight of the coupling from j^{th} p-bit to i^{th} p-bit, I_0 parameterizes the coupling strength between p-bits, and τ_S is the synapse evaluation time. Several hardware designs of p-bits based on low barrier nanomagnet (LBM) physics have been proposed and also experimentally demonstrated (Ostwal et al., 2018; Borders et al., 2019; Ostwal and Appenzeller, 2019; Camsari et al., 2020; Debashis, 2020). The thermal energy barrier of the LBM is of the order of a few $k_B T$ instead of $40\text{--}60 k_B T$ used in the memory technology to retain stability. Because of thermal noise the magnetization of the LBM keeps fluctuating as a function of time with an average retention time $\tau \sim \tau_0 \exp(E_B/k_B T)$ (Brown, 1979), where τ_0 is a material-dependent parameter called attempt time that is experimentally found to be in the range of nanosecond or less and

E_B is the thermal energy barrier (Lopez-Diaz et al., 2002; Pufall et al., 2004). The stochasticity of the LBMs makes them naturally suitable for p-bit implementation.

Figure 1 shows two p-bit designs: Design 1 (**Figure 1B**) (Camsari et al., 2017b; Borders et al., 2019) and Design 2 (**Figure 1C**) (Camsari et al., 2017a; Ostwal and Appenzeller, 2019). Designs 1 and 2 both are fundamental building blocks of spin transfer torque (STT) and spin orbit torque (SOT) magnetoresistive random access memory (MRAM) technologies, respectively (Bhatti et al., 2017). Their technological relevance motivates us to explore their implementations as p-bits. Design 1 is very similar to the commercially available 1T/1MTJ (T: Transistor, MTJ: Magnetic Tunnel Junction) embedded MRAM device where the free layer of the MTJ is replaced by an in-plane magnetic anisotropy (IMA) or perpendicular magnetic anisotropy (PMA) LBM. Design 2 is similar to the basic building block of SOT-MRAM device (Liu et al., 2012) where the thermal fluctuation of the free layer magnetization of the stochastic MTJ (s-MTJ) (Vodenicarevic et al., 2017, 2018; Mizrahi et al., 2018; Parks et al., 2018; Zink et al., 2018; Borders et al., 2019) is tuned by a spin current generated in a heavy metal layer underneath the LBM due to SOT effect. The in-plane polarized spin current from the SOT effect in the spin hall effect (SHE) material in design 2 requires an in-plane LBM to tune its magnetization, although a perpendicular LBM with a tilted anisotropy axis is also experimentally shown to work (Debashis et al., 2020). However, design 2 requires spin current manipulation, design 1 does not rely on that as long as circular in-plane LBMs with continuous valued magnetization states that are hard to pin are used. In-plane LBMs also provide faster fluctuation than perpendicular ones leading to faster sampling speed in the probabilistic hardware (Hassan et al., 2019; Kaiser et al., 2019).

The key distinguishing feature of the two p-bit designs (designs 1 and 2) is the time scales in implementing Equation (1a). From a hardware point of view, Equation (1a) has two components: a random number generator (RNG) (rand) and a tunable component (\tanh). In design 1, the RNG is the s-MTJ utilizing an LBM and the tunable component is the NMOS transistor, thus having two different time scales in the equation. But in design 2, both the RNG and the tunable component are implemented by a single s-MTJ utilizing an LBM, thus having just one time scale in the equation. This difference in time scales in the two designs is shown in **Figure 2**. Note that although the two p-bit designs have the same RNG source, namely a fluctuating magnetization, it is the difference in their circuit configuration with or without the NMOS transistor in the MTJ branch that results in different time dynamics of the two designs.

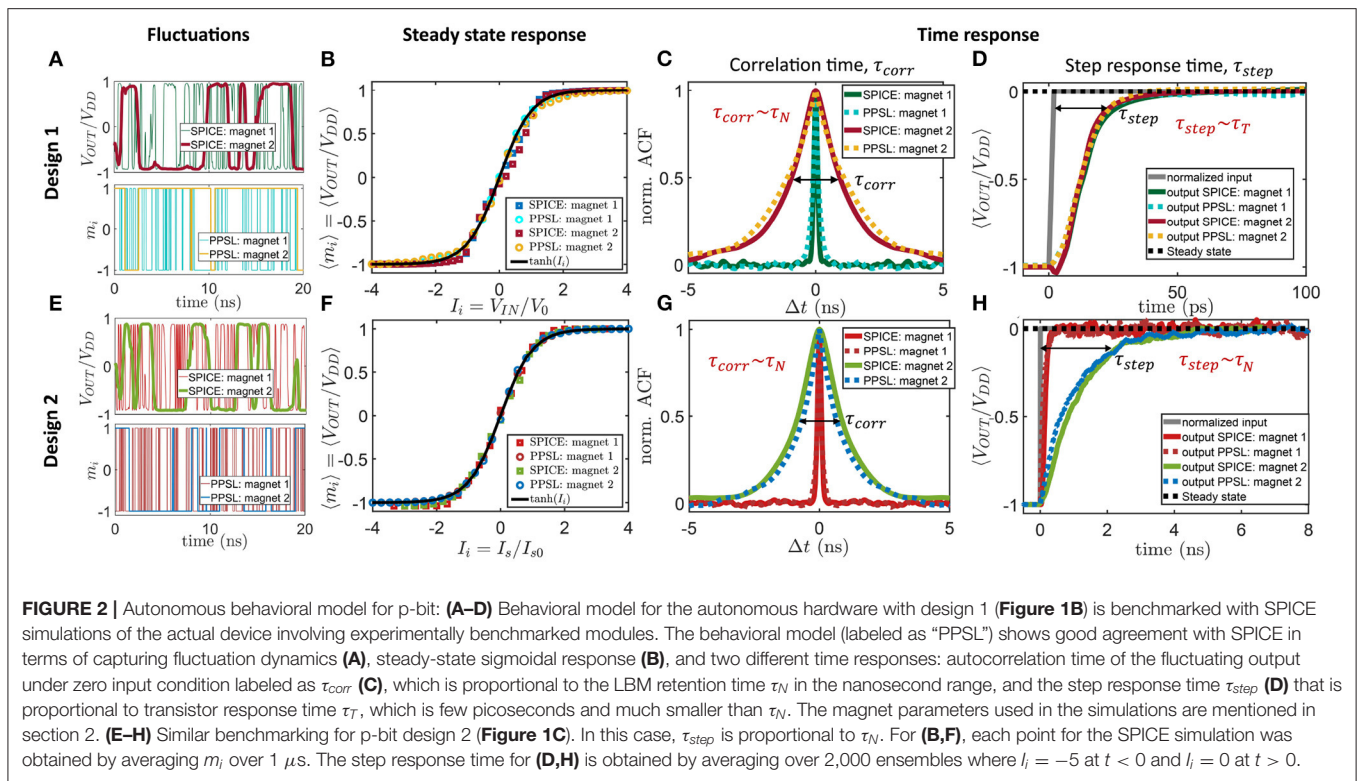
In traditional software implementations, p-bits are updated sequentially for accurate operation such that after each $\tau_S + \tau_N$ time interval, only one p-bit is updated (Hinton, 2007). This naturally implies the use of sequencers to ensure the sequential update of p-bits. The sequencer generates an *Enable* signal for each p-bit in the network and ensures that no two p-bits update simultaneously. The sequencer also makes sure that every p-bit is updated at least once in a time step where each time step corresponds to $N \cdot (\tau_S + \tau_N)$, N being the number of p-bits in the network. (Roberts and Sahu, 1997; Pervaiz et al.,



2018). For symmetrically connected networks ($J_{ij} = J_{ji}$) such as Boltzmann machines, the update order of p-bits does not matter and any random update order produces the standard probability distribution described by equilibrium Boltzmann law as long as p-bits are updated sequentially. But for directed acyclic networks ($J_{ij} \neq 0, J_{ji} = 0$) or BNs to be consistent with the expected conditional probability distribution, p-bits need to be updated not only sequentially but also in a specific update order, which is from the parent to child nodes (Neal, 1992) similar to the concept of forward sampling in belief networks (Henrion,

1988; Guo and Hsu, 2002; Koller and Friedman, 2009). As long as this parent to child update order is maintained, the network converges to the correct probability distribution described by probability chain rule or Bayes rule. This effect of update order in a sequential p-circuit is shown on a three p-bit network in **Figure 1D**. In the **Supplementary Material**, it is shown in an example how the CPT of the BN can be mapped to a p-circuit following Faria et al. (2018).

Unlike sequential p-circuits in ANN literature, the distinguishing feature of our probabilistic hardware is that



it is *autonomous* where each p-bit runs in parallel without any clocks or sequencers. This autonomous p-circuit (ApC) allows massive parallelism potentially providing peta flips per second sampling speed (Sutton et al., 2020). The complete sequencer-free operation of our “autonomous” p-circuit is very different from the “asynchronous” operation of spiking neural networks (Merolla et al., 2014; Davies et al., 2018). Although p-bits are fluctuating in parallel in an ApC, it is very unlikely that two p-bits will update at the exact same time since random noise control their dynamics. Therefore, persistent parallel updates are extremely unlikely and are not a concern. Note that even if p-bits update sequentially, each update has to be *informed* such that when one p-bit updates it has received the up-to-date input I_i based on the latest states of other p-bits m_j that it is connected to. This informed update can be ensured as long as the synapse response time is much faster than the neuron time ($\tau_S \ll \tau_N$) and this is a key design rule for an ApC. If the input of the p-bit is based on old state of neighboring p-bits or on time-integrated synaptic inputs, the ApC operation declines in functionality or fails completely. However, for $\tau_S \ll \tau_N$, the ApC works properly for a Boltzmann network without any clock because no specific update order is required in this case. But, it is not intuitive at all if an ApC would work for a BN because a particular parent to child *informed* update order is required in this case, as shown in Figure 1D. As such, it is not straightforward that a clockless autonomous circuit can naturally ensure this specific informed update order. In Figure 1E, we have shown that it is possible to design hardware p-circuit that can naturally ensure a parent to child informed update order in a BN without any clocks. In

Figure 1E, two p-bit designs are evaluated for implementing both Boltzmann network and BN. We have shown that design 1 is suitable for both Boltzmann network and BN. But design 2 is suitable for Boltzmann networks only and does not work for BNs in general. The synapse in both types of p-circuits is implemented using a resistive crossbar architecture (Alibart et al., 2013; Camsari et al., 2017b), although there are also other types of hardware synapse implementations based on memristors (Li et al., 2018; Mahmoodi et al., 2019; Mansueto et al., 2019), magnetic tunnel junctions (Ostwal et al., 2019), spin orbit torque driven domain wall motion devices (Zand et al., 2018), phase change memory devices (Ambrogio et al., 2018), and so on. In all the simulations, τ_S is assumed to be negligible compared to other time scales in the circuit dynamics.

Our proposed probabilistic hardware for BNs shows significant biological relevance because of the following reasons: (1) The brain consists of neurons and synapses. The basic building block called “p-bit” of our proposed hardware mimics the neuron and the interconnection among p-bits mimics the synapse function. (2) The components of brain are stochastic or noisy by nature. p-bits mimicking the neural dynamics in our proposed hardware are also stochastic. (3) Brain does not have a single clock for synchronous operation and can perform massively parallel processing (Strukov et al., 2019). Our autonomous hardware also does not have any global clock or sequencers and each p-bit fluctuates in parallel allowing massively parallel operation.

Further, we have provided a behavioral model in section 2 for both designs 1 and 2, illustrating the essential characteristics

needed for correct sequencer-free operation of BNs. Both models are benchmarked against state-of-the-art device/circuit models (SPICE) of the actual devices and can be used for the efficient simulation of large-scale autonomous networks.

2. BEHAVIORAL MODEL FOR AUTONOMOUS HARDWARE

In this section, we will develop an autonomous behavioral model that we will call parallel probabilistic spin logic (PPSL) for design 1 (**Figure 1B**) and revisit the behavioral model for design 2, which was proposed by Sutton et al. (2020). The term “Parallel” refers to all the p-bits fluctuating in parallel without any clocks or sequencers. These behavioral models are high-level representations of the p-circuit and p-bit behavior and connect Equations (1a) and (1b) to the hardware p-bit designs. Please note the parameters introduced in these models will represent certain parts of the p-bit and synapse behavior like MTJ resistances (r_{MTJ}) and transistor resistances (r_T) but are generally dimensionless apart from time variables (e.g., τ_T, τ_N). The advantage of these models is that they are computationally less expensive to use than full SPICE simulations while preserving the crucial device and system characteristics.

2.1. Autonomous Behavioral Model: Design 1

The autonomous circuit behavior of design 1 can be explained by slightly modifying the two equations (Equations 1a,b) stated in section 1. The fluctuating resistance of the low barrier nanomagnet-based MTJ is represented by a correlated random number r_{MTJ} with values between -1 and $+1$ and an average dwell time of the fluctuation denoted by τ_N . The NMOS transistor tunable resistance is denoted by r_T and the inverter is represented by a sgn function. Thus, the normalized output $m_i = V_{OUT,i}/V_{DD}$ of the i_{th} p-bit can be expressed as:

$$m_i(t + \Delta t) = \text{sgn}(r_{T,i}(t + \Delta t) - r_{MTJ,i}(t + \Delta t)) \quad (2)$$

where Δt is the simulation time step, $r_{T,i}$ represents the NMOS transistor resistance tunable by the normalized input $I_i = V_{IN,i}/V_0$ (compare Equation 1a) where V_0 is a fitting parameter which is $\approx 50\text{mV}$ for the chosen parameters and transistor technology (compare **Figure 2B**) and $r_{MTJ,i}$ is a correlated random number generator with an average retention time of τ_N . For design 1, the transistor represents the tunable component that works in conjunction with the unbiased stochastic signal of the MTJ. $r_{T,i}$ as a function of input I_i is approximated by a tanh function with a response time denoted by τ_T modeled by the following equations:

$$r_{T,i}(t + \Delta t) = r_{T,i}(t) \exp(-\Delta t/\tau_T) + (1 - \exp(-\Delta t/\tau_T)) (\tanh(I_i(t + \Delta t))) \quad (3)$$

where it can be clearly seen that the dimensionless quantity $r_{T,i}$ representing the transistor resistance is bounded by $-1 \leq r_{T,i} \leq 1$ for all synaptic inputs. **Figure 2B** shows by utilizing SPICE simulation how I_i influences the average output m_i and shows

that the average response of the circuit is in good agreement with the tanh-function used in Equation (3).

The synapse delay τ_S in computing the input I_i can be modeled by:

$$I_i(t + \Delta t) = I_i(t) \exp(-\Delta t/\tau_S) + (1 - \exp(-\Delta t/\tau_S)) \left(I_0 \left(\sum_j I_{ij} m_j(t) + h_j \right) \right) \quad (4)$$

For calculating $r_{MTJ,i}$ at time $t + \Delta t$ a new random number will be picked according to the following equations:

$$r_{flip,i}(t + \Delta t) = \text{sgn} \left(\exp \left(-\frac{\Delta t}{\tau_N} \right) - \text{rand}_{[0,1]} \right) \quad (5a)$$

where $\text{rand}_{[0,1]}$ is a uniformly distributed random number between 0 and 1 and τ_N represents the average retention time of the fluctuating MTJ resistance. If r_{flip} is -1 , a new random r_{MTJ} will be chosen between -1 and $+1$. Otherwise, the previous $r_{MTJ}(t)$ will be kept in the next time step ($t + \Delta t$), which can be expressed as

$$r_{MTJ,i}(t + \Delta t) = \frac{r_{flip,i}(t + \Delta t) + 1}{2} r_{MTJ,i}(t) - \frac{r_{flip,i}(t + \Delta t) - 1}{2} \text{rand}_{[-1,1]} \quad (5b)$$

where $-1 \leq r_{MTJ,i}(t) \leq 1$.

The charge current flowing through the MTJ branch of p-bit design 1 can get polarized by the fixed layer of the MTJ and generate a spin current I_s that can tune/pin the MTJ dynamics by modifying τ_N . This effect is needed for tuning the output of design 2 but is not desired in design 1. However, the developed behavioral model can account for this pinning effect according to

$$\tau_N = \tau_N^0 \exp(r_{MTJ} I_{MTJ}), \quad (6)$$

where τ_N^0 is the retention time of r_{MTJ} when $I_{MTJ} = 0$. The dimensionless pinning current I_{MTJ} is defined as $I_{MTJ} = I_s/I_{s,0}$ where $I_{s,0}$ can be extracted by following the procedure of **Figure 2F**. This pinning effect by I_{MTJ} is much smaller in in-plane magnets (IMA) than perpendicular magnets (PMA) (Hassan et al., 2019) and is ignored for design 1 throughout this paper.

Figures 2A–D shows the comparison of this behavioral model for p-bit design 1 with SPICE simulation of the actual hardware in terms of fluctuation dynamics, sigmoidal characteristic response, autocorrelation time (τ_{corr}), and step response time (τ_{step}) and in all cases the behavioral model closely matches SPICE simulations. The SPICE simulation involves experimentally benchmarked modules for different parts of the device. The SPICE model for the s-MTJ model solves the stochastic Landau–Lifshitz–Gilbert equation for the LBM physics. For the transistors, 14 nm Predictive Technology Model¹ is used. As simulator *HSPICE*

¹<http://ptm.asu.edu/>

is utilized with the `.trannoise` function and a time step of 1 ps. The simulating framework was benchmarked experimentally and by using standard simulation tools in the field (Datta, 2012; Torunbalci et al., 2018). The autonomous behavioral model for design 1 is labeled as “PPSL: design 1.” The benchmarking is done for two different LBMs: (1) Faster fluctuating magnet 1 with saturation magnetization $M_s = 1100$ emu/cc, diameter $D = 22$ nm, thickness $th = 2$ nm, in-plane easy axis anisotropy $H_k = 1$ Oe, damping coefficient $\alpha = 0.01$, demagnetization field $H_d = 4\pi M_s$ and (2) slower fluctuating magnet 2 with the same parameters as in magnet 1 except $D = 150$ nm. The supply voltage was set to $V_{DD} = -V_{SS} = 0.4$ V. The fast and slow fluctuations of the normalized output $m_i = V_{OUT,i}/V_{DD}$ are captured by changing the τ_N parameter in the PPSL model. In the steady-state sigmoidal response, V_0 is a tanh fitting parameter that defines the width of the sigmoid and lies within the range of 40–60 mV reasonably well depending on which part of the sigmoid needs to be better matched. In **Figure 2B**, V_0 value of 50 mV is used to fit the sigmoid from SPICE simulation. The following parameters have been extracted from the calibration shown in **Figure 2**, where $\Delta t = 1$ ps was used: $\tau_N = 150$ ps (magnet 1), $\tau_N = 1.5$ ns (magnet 2), $\tau_T = 3$ ps, $I_{s,0} = 120$ μ A (magnet 1), $I_{s,0} = 1$ mA (magnet 2).

There are two types of time responses: (1) Autocorrelation time under zero input condition labeled as τ_{corr} and (2) step response time τ_{step} . The full width half maximum (FWHM) of the autocorrelation function of the fluctuating output under zero input is defined by τ_{corr} , which is proportional to the retention time τ_N of the LBM. The step response time τ_{step} is obtained by taking an average of the p-bit output over many ensembles when the input I_i is stepped from a large negative value to zero at time $t = 0$ and measuring the time it takes for the ensemble averaged output to reach its statistically correct value consistent with the new input. τ_{step} defines how fast the first statistically correct sample can be obtained after the input is changed. For p-bit design 1, τ_{step} is independent of LBM retention time τ_N and is defined by the NMOS transistor response time τ_T , which is much faster (few picoseconds) than LBM fluctuation time τ_N . The effect of these two very different time scales in design 1 ($\tau_{step} \ll \tau_{corr}$) on an autonomous BN is described in section 3.

2.2. Autonomous Behavioral Model: Design 2

The autonomous behavioral model for design 2 is proposed in Sutton et al. (2020). In this article, we have benchmarked this model with the SPICE simulation of the single p-bit steady state and time responses shown in **Figures 2E–H**. According to this model, the normalized output $m_i = V_{OUT,i}/V_{DD}$ can be expressed as:

$$m_i(t + \Delta t) = m_i(t) \text{sgn}\left(p_{NOTflip,i}(t + \Delta t) - \text{rand}_{[0,1]}\right) \quad (7a)$$

$$p_{NOTflip,i}(t + \Delta t) = \exp\left(-\frac{\Delta t}{\tau_N \exp(I_i m_i(t))}\right) \quad (7b)$$

where $p_{NOTflip,i}(t + \Delta t)$ is the probability of retention of the i^{th} p-bit (or “not flipping”) in the next time step that

is a function of average neuron flip time τ_N , input I_i , and the current p-bit output $m_i(t)$. **Figure 2** shows how this simple autonomous behavioral model for design 2 matches reasonably well with SPICE simulation of the device in terms of fluctuation dynamics (**Figure 2E**), sigmoidal characteristic response (**Figure 2F**), autocorrelation time (τ_{corr}) (**Figure 2G**), and step response time (τ_{step}) (**Figure 2H**). In design 2, τ_{step} and τ_{corr} are both proportional to LBM fluctuation time τ_N unlike design 1.

Different time scales in p-bit designs 1 and 2 are also reported in Hassan et al. (2019) in an energy-delay analysis context. In this article, we explain the effect of these time scales in designing an autonomous BN (section 3).

3. DIFFERENCE BETWEEN DESIGNS 1 AND 2 IN IMPLEMENTING BAYESIAN NETWORKS

The behavioral models introduced in section 2 are applied to implement a multi-layer belief/BN with 19 p-bits and random interconnection strengths between +1 and −1 (**Figure 3A**). For illustrative purposes, the interconnections are designed in such a way that although there are no meaningful correlations between the blue and red colored nodes with random couplings, pairs of intermediate nodes (A, M_1) and (M_1, B) get negatively correlated because of a net $-r^2$ type coupling through each branch connecting the pairs. So it is expected that the start and end nodes (A, B) get positively correlated. **Figure 3B** shows histograms of four configurations (00, 01, 10, 11) of the pair of nodes A and B obtained from different approaches: Bayes rule (labeled as Analytic), SPICE simulation of design 1 (SPICE: Design 1) and design 2 (SPICE: Design 2), and autonomous behavioral model for design 1 (PPSL: Design 1) and design 2 (PPSL: design 2). It is shown that results from SPICE simulation and behavioral model for design 1 matches reasonably well with the standard analytical values showing 00 and 11 states with highest probability, whereas design 2 autonomous hardware does not work well in terms of matching with the analytical results and shows approximately all equal peaks. We have tested this basic conclusion for other networks as well with more complex topology as shown in **Supplementary Figure 1**. The analytical values are obtained from applying the standard joint probability rule for BNs (Pearl, 2014; Russell and Norvig, 2016), which is:

$$P(x_1, x_2, \dots, x_N) = \prod_{i=1}^N P(x_i | \text{Parents}(x_i)) \quad (8)$$

Joint probability between two specific nodes x_i and x_j can be calculated from the above equation by summing over all configurations of the other nodes in the network, which becomes computationally expensive for larger networks. But one major advantage of our probabilistic hardware is that probabilities of specific nodes can be obtained by looking at the nodes of interest ignoring all other nodes in the system similar to what Feynman stated about a probabilistic computer imitating the probabilistic laws of nature (Feynman, 1982). Indeed, in the BN example in

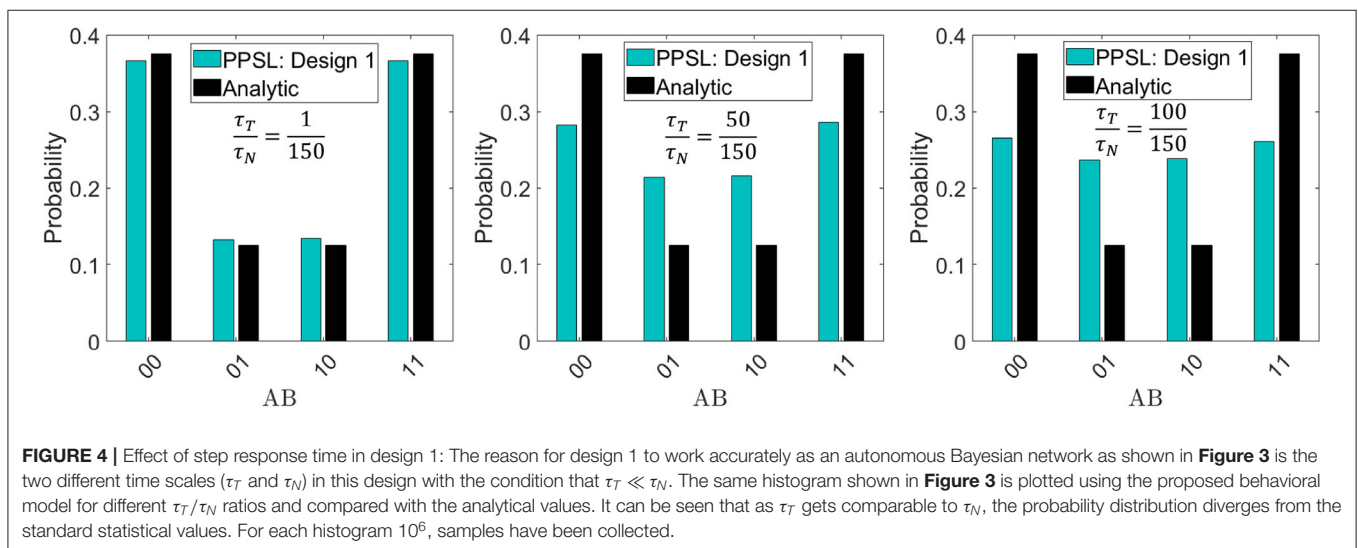
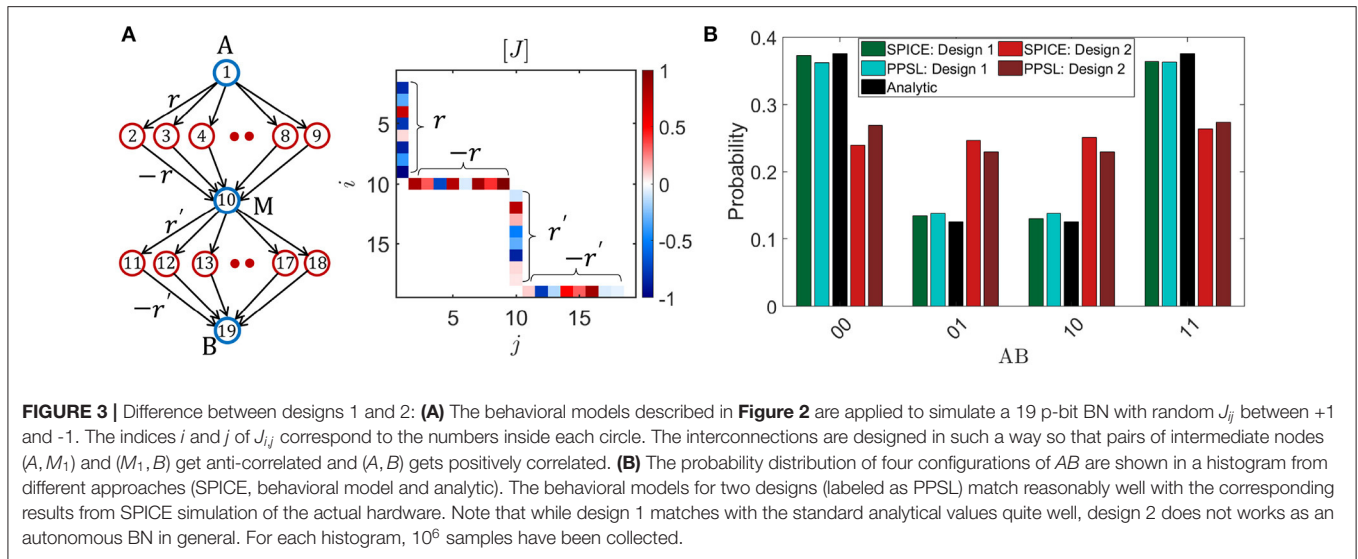


Figure 3, the probabilities of different configurations of nodes A and B were obtained by looking at the fluctuating outputs of the two nodes ignoring all other nodes. For the SPICE simulation of design 1 hardware, tanh fitting parameter $V_0 = 57$ mV is used and the mapping principle from dimensionless coupling terms J_{ij} to the coupling resistances in the hardware is described in Faria et al. (2018). An example of this mapping is given in the **Supplementary Material**.

The reason why design 1 works for a BN and design 2 does not is because of the two very different time responses of the two designs shown in **Figure 2** due to the fact that the tunable component is the transistor in design 1 ($\tau_{step} \propto \tau_T$) and the MTJ in design 2 ($\tau_{step} \propto \tau_N$). It is these two different time scales in design 1 ($\tau_{step} \ll \tau_{corr}$) that naturally ensures a parent to child informed update order in a BN. The reason is that when τ_{step} is small, each child node can immediately respond to any change of its parent nodes that happens due to a random event, which have a much larger time scale $\propto \tau_{corr}$. Thus, due to that fast step response, information about changing p-bits at the parent

node can propagate quickly through the network and the output of the child nodes can be conditionally satisfied with the parent nodes very fast. Otherwise, if τ_{corr} gets comparable to τ_{step} , the child nodes will not be able to keep up with the fast changing parent nodes since the information of the parent p-bit state has not been propagated through the network. As a result, the child nodes will produce a substantial number of statistically incorrect samples over the entire time range, thus deviating from the correct probability distribution. This effect is especially strong for networks where the coupling strength between p-bits is large.

To illustrate this point, the effect of τ_{step}/τ_{corr} ratio is shown in **Figure 4** for the same BN presented in **Figure 3** by plotting the histogram of AB configurations for different τ_T/τ_N ratios. It is shown that when τ_T/τ_N ratio is small, the histogram converges to the correct distribution. As τ_T gets comparable to τ_N , the histogram begins to diverge from the correct distribution. Thus, the very fast NMOS transistor response in design 1 makes it suitable for an autonomous BN hardware. One thing to note that under certain conditions, results from design 2 can also match

the analytical results if spin current bias is large enough to drive down the fast step response time to ensure $\tau_{step} \ll \tau_{corr}$.

So apart from ensuring a fast synapse compared to neuron fluctuation time ($\tau_S \ll \tau_N$), which is the design rule for an autonomous probabilistic hardware, the autonomous BN demands an additional p-bit design rule that is a much faster step response time of the p-bit compared to its fluctuation time ($\tau_{step} \ll \tau_N$) as ensured in design 1. In all the simulations, the LBM was a circular in-plane magnet whose magnetization spans all values between +1 and -1 and negligible pinning effect. If the LBM is a PMA magnet with bipolar fluctuations having just two values +1 and -1, design 1 will not provide any sigmoidal response except with substantial pinning effect (Borders et al., 2019). Under this condition, τ_{step} of design 1 will be comparable to τ_N again and the system will not work as an autonomous BN in general. Therefore, LBM with continuous range fluctuation is expected for design 1 p-bit to work properly as a BN.

4. DISCUSSION

In this article, we have elucidated the design criteria for an autonomous clockless hardware for BNs that requires a specific parent to child update order when implemented on a probabilistic circuit. By performing SPICE simulations of two autonomous probabilistic hardware designs built out of p-bits (designs 1 and 2 in **Figure 1**), we have shown that the autonomous hardware will naturally ensure a parent to child informed update order without any sequencers if the step response time (τ_{step}) of the p-bit is much smaller than its autocorrelation time (τ_{corr}). This criteria of having two different time scales is met in design 1 as τ_{step} comes from the NMOS transistor response time τ_T in this design, which is few picoseconds. We have also proposed an autonomous behavioral model for design 1 and benchmarked it against SPICE simulation of the actual hardware. All the simulations using behavioral model for design 1 are performed ignoring some non-ideal effects listed as follows:

- Pinning of the s-MTJ fluctuation due to STT effect is ignored by assuming $I_{MTJ} = 0$ in Equation (6). This is a reasonable assumption considering circular in-plane magnets that are very difficult to pin due to the large demagnetization field that is always present, irrespective of the energy barrier (Hassan et al., 2019). This effect is more prominent in perpendicular anisotropy magnets (PMA) magnets. It is important to include the pinning effect in p-bits with bipolar LBM fluctuations because in this case the p-bit does not provide a sigmoidal response without the pinning current. This effect is also experimentally observed in Borders et al. (2019) for PMA magnets. Such a p-bit design with bipolar PMA and STT pinning might not work for BNs in general, because in this case τ_{step} will be dependent on magnet fluctuation time τ_N .
- In the proposed behavioral model, the step response time of the NMOS transistor τ_T in design 1 is assumed to be independent of the input I . But there is a functional dependence of τ_T on I in real hardware.
- The NMOS transistor resistance r_T is approximated as a tanh function for simplicity. In order to capture the hardware behavior in a better way, the tanh can be replaced by a more complicated function and the weight matrix $[J]$ will have to be learnt around that function.

All the non-ideal effects listed above are supposed to have minimal effects on different probability distributions shown in this article. Real LBMs may suffer from common fabrication defects, resulting in variations in average magnet fluctuation time τ_N (Abeed and Bandyopadhyay, 2019). The autonomous BN is also quite tolerant to such variations in τ_N as long as $\tau_T \ll \min(\tau_N)$.

It is important to note that, for design 1 (Transistor-controlled) to function as a p-bit that has a step response time (τ_{step}) much smaller than its average fluctuation time (τ_N), the LBM fluctuation needs to be continuous and not bipolar. It is important to note that while most experimental implementations of low barrier magnetic tunnel junctions or spin-valves exhibit telegraphic (binary) fluctuations (Pufall et al., 2004; Locatelli et al., 2014; Parks et al., 2018; Debashis et al., 2020), theoretical results (Abeed and Bandyopadhyay, 2019; Hassan et al., 2019; Kaiser et al., 2019) indicate that it should be possible to design low barrier magnets with continuous fluctuations. Preliminary experimental results for such circular disk nanomagnets have been presented in Debashis et al. (2016). We believe that a lack of experimental literature on such magnets is partly due to the lack of interest of randomly fluctuating magnets that have long been discarded as impractical and irrelevant. The other experimentally demonstrated p-bits (Ostwal et al., 2018; Ostwal and Appenzeller, 2019; Debashis, 2020) fall under design 2 category with the LBM magnetization tuned by SOT effect and are not suitable for autonomous BN operation in general. It might also be possible to design p-bits using other phenomena such as voltage controlled magnetic anisotropy (Amiri and Wang, 2012), but this is beyond the scope of the present study. Here, we have specifically focused on two designs that can be implemented with existing MRAM technology based on STT and SOT.

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

RF and SD wrote the paper. RF performed the simulations. JK helped setting up the simulations. KC developed the simulation modules for the BSN in SPICE. 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.

ACKNOWLEDGMENTS

The authors would like to thank professor Joerg Appenzeller from Purdue University for helpful discussions. This manuscript has been released as a preprint at arXiv (Faria et al., 2020).

REFERENCES

- 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 Magnet. 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. doi: 10.1207/s15516709cog0901_7
- Alibart, F., Zamanidoost, E., and Strukov, D. B. (2013). Pattern classification by memristive crossbar circuits using *ex situ* and *in situ* training. *Nat. Commun.* 4, 1–7. doi: 10.1038/ncomms3072
- 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
- Amiri, P. K., and Wang, K. L. (2012). “Voltage-controlled magnetic anisotropy in spintronic devices,” in *Spin*, Vol. 2 (World Scientific), 1240002. doi: 10.1142/S2010324712400024
- Arias, J., Martinez-Gomez, J., Gamez, J. A., de Herrera, A. G. S., and Müller, H. (2016). Medical image modality classification using discrete bayesian networks. *Comput. Vis. Image Understand.* 151, 61–71. doi: 10.1016/j.cviu.2016.04.002
- Behin-Aein, B., Diep, V., and Datta, S. (2016). A building block for hardware belief networks. *Sci. Rep.* 6:29893. doi: 10.1038/srep29893
- Bhatti, S., Sbiaa, R., Hirohata, A., Ohno, H., Fukami, S., and Piramanayagam, S. (2017). Spintronics based random access memory: a review. *Mater. Tdy.* 20, 530–548. doi: 10.1016/j.mattod.2017.07.007
- Bielza, C., and Larrañaga, P. (2014). Bayesian networks in neuroscience: a survey. *Front. Comput. Neurosci.* 8:131. doi: 10.3389/fncom.2014.00131
- 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
- Brown, W. (1979). Thermal fluctuation of fine ferromagnetic particles. *IEEE Trans. Magnet.* 15, 1196–1208. doi: 10.1109/TMAG.1979.1060329
- Camsari, K. Y., Chowdhury, S., and Datta, S. (2019). Scalable emulation of sign-problem-free hamiltonians with room-temperature *p*-bits. *Phys. Rev. Appl.* 12:034061. doi: 10.1103/PhysRevApplied.12.034061
- Camsari, K. Y., Debashis, P., Ostwal, V., Pervaiz, A. Z., Shen, T., Chen, Z., et al. (2020). From charge to spin and spin to charge: Stochastic magnets for probabilistic switching. *Proc. IEEE*. 108:1322–1337. doi: 10.1109/JPROC.2020.2966925
- 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., Salahuddin, S., and Datta, S. (2017b). Implementing *p*-bits with embedded mtj. *IEEE Electron Dev. Lett.* 38, 1767–1770. doi: 10.1109/LED.2017.2768321
- Chakrapani, L. N., Korkmaz, P., Akgul, B. E., and Palem, K. V. (2007). Probabilistic system-on-a-chip architectures. *ACM Trans. Design Automat. Electron. Syst.* 12:29. doi: 10.1145/1255456.1255466
- Correa, M., Bielza, C., and Pamiès-Teixeira, J. (2009). Comparison of bayesian networks and artificial neural networks for quality detection in a machining process. *Expert Syst. Appl.* 36, 7270–7279. doi: 10.1016/j.eswa.2008.09.024
- Darwiche, A. (2009). *Modeling and Reasoning With Bayesian Networks*. Cambridge University Press. Available online at: <https://www.cambridge.org/core/books/modeling-and-reasoning-with-bayesian-networks/8A3769B81540EA93B525C4C2700C9DE6> doi: 10.1017/CBO9780511811357
- Datta, D. (2012). *Modeling of spin transport in MTJ devices* (Ph.D. thesis), Purdue University Graduate School; ProQuest. Available online at: <https://docs.lib.purdue.edu/dissertations/AA13556189/>
- Davies, M., Srinivasa, N., Lin, T.-H., China, 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
- Debashis, P. (2020). *Spintronic devices as P-bits for probabilistic computing* (Ph.D. thesis). Purdue University Graduate School. Available online at: https://hammer.figshare.com/articles/thesis/Spintronic_Devices_as_P-bits_for_Probabilistic_Computing/11950395
- Debashis, P., Faria, R., Camsari, K. Y., Appenzeller, J., Datta, S., and Chen, Z. (2016). “Experimental demonstration of nanomagnet networks as hardware for ising computing,” in *2016 IEEE International Electron Devices Meeting (IEDM)* (San Francisco, CA: IEEE), 34. doi: 10.1109/IEDM.2016.7838539
- Debashis, P., Faria, R., Camsari, K. Y., Datta, S., and Chen, Z. (2020). Correlated fluctuations in spin orbit torque coupled perpendicular nanomagnets. *Phys. Rev. B* 101:094405. doi: 10.1103/PhysRevB.101.094405
- Faria, R., Camsari, K. Y., and Datta, S. (2017). Low-barrier nanomagnets as *p*-bits for spin logic. *IEEE Magnet. Lett.* 8, 1–5. doi: 10.1109/LMAG.2017.2685358
- Faria, R., Camsari, K. Y., and Datta, S. (2018). Implementing bayesian networks with embedded stochastic mram. *AIP Adv.* 8:045101. doi: 10.1063/1.5021332
- Faria, R., Kaiser, J., Camsari, K. Y., and Datta, S. (2020). Hardware design for autonomous bayesian networks. *arXiv preprint arXiv:2003.01767*.
- Feynman, R. P. (1982). Simulating physics with computers. *Int. J. Theor. Phys.* 21:467–488. doi: 10.1007/BF02650179
- Friedman, J. S., Calvet, L. E., Bessi re, P., Droulez, J., and Querlioz, D. (2016). Bayesian inference with muller *c*-elements. *IEEE Trans. Circ. Syst. I Regular Pap.* 63, 895–904. doi: 10.1109/TCSI.2016.2546064
- Friedman, N., Linial, M., Nachman, I., and Pe r, D. (2000). Using bayesian networks to analyze expression data. *J. Comput. Biol.* 7, 601–620. doi: 10.1089/106652700750050961
- Guo, H., and Hsu, W. (2002). “A survey of algorithms for real-time bayesian network inference,” in *Join Workshop on Real Time Decision Support and Diagnosis Systems*. American Association for Artificial Intelligence. Available online at: <https://www.aaai.org/Papers/Workshops/2002/WS-02-15/WS02-15-001.pdf>
- Hassan, O., Faria, R., Camsari, K. Y., Sun, J. Z., and Datta, S. (2019). Low-barrier magnet design for efficient hardware binary stochastic neurons. *IEEE Magnet. Lett.* 10, 1–5. doi: 10.1109/LMAG.2019.2910787
- Heckerman, D., and Breese, J. S. (1996). Causal independence for probability assessment and inference using bayesian networks. *IEEE Trans. Syst. Man Cybernet A Syst. Hum.* 26, 826–831. doi: 10.1109/3468.541341
- Henrion, M. (1988). “Propagating uncertainty in bayesian networks by probabilistic logic sampling,” in *Machine Intelligence and Pattern Recognition*, (Elsevier) 5:149–163. doi: 10.1016/B978-0-444-70396-5.50019-4
- Hinton, G. E. (2007). Boltzmann machine. *Scholarpedia* 2:1668. doi: 10.4249/scholarpedia.1668
- Jansen, R., Yu, H., Greenbaum, D., Kluger, Y., Krogan, N. J., Chung, S., et al. (2003). A bayesian networks approach for predicting protein-protein interactions from genomic data. *Science* 302, 449–453. doi: 10.1126/science.1087361
- Jonas, E. M. (2014). *Stochastic architectures for probabilistic computation* (Ph.D. thesis). Massachusetts Institute of Technology, Cambridge, MA, United States.
- Kaiser, J., Faria, R., Camsari, K. Y., and Datta, S. (2020). Probabilistic circuits for autonomous learning: a simulation study. *Front. Comput. Neurosci.* 14:14. doi: 10.3389/fncom.2020.00014
- Kaiser, J., Rustagi, A., Camsari, K., Sun, J., Datta, S., and Upadhyaya, P. (2019). Subnanosecond fluctuations in low-barrier nanomagnets. *Phys. Rev. Appl.* 12:054056. doi: 10.1103/PhysRevApplied.12.054056
- Koller, D., and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. Available online at: <https://mitpress.mit.edu/books/probabilistic-graphical-models>
- Li, C., Belkin, D., Li, Y., Yan, P., Hu, M., Ge, N., et al. (2018). Efficient and self-adaptive *in-situ* learning in multilayer memristor neural networks. *Nat. Commun.* 9, 1–8. doi: 10.1038/s41467-018-04484-2
- Liu, L., Pai, C.-F., Li, Y., Tseng, H., Ralph, D., and Buhrman, R. (2012). Spin-torque switching with the giant spin hall effect of tantalum. *Science* 336, 555–558. doi: 10.1126/science.1218197
- Locatelli, N., Mizrahi, A., Accioly, A., Matsumoto, R., Fukushima, A., Kubota, H., et al. (2014). Noise-enhanced synchronization of stochastic magnetic oscillators. *Phys. Rev. Appl.* 2:034009. doi: 10.1103/PhysRevApplied.2.034009

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fncom.2021.584797/full#supplementary-material>

- Lopez-Diaz, L., Torres, L., and Moro, E. (2002). Transition from ferromagnetism to superparamagnetism on the nanosecond time scale. *Phys. Rev. B* 65:224406. doi: 10.1103/PhysRevB.65.224406
- Mahmoodi, M., Prezioso, M., and Strukov, D. (2019). Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization. *Nat. Commun.* 10, 1–10. doi: 10.1038/s41467-019-13103-7
- Mansueto, M., Chavent, A., Auffret, S., Joumard, I., Nath, J., Miron, I. M., et al. (2019). Realizing an isotropically coercive magnetic layer for memristive applications by analogy to dry friction. *Phys. Rev. Appl.* 12:044029. doi: 10.1103/PhysRevApplied.12.044029
- 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
- Mizrahi, A., Hirtzlin, T., Fukushima, A., Kubota, H., Yuasa, S., Grollier, J., et al. (2018). Neural-like computing with populations of superparamagnetic basis functions. *Nat. Commun.* 9, 1–11. doi: 10.1038/s41467-018-03963-w
- Neal, R. M. (1992). Connectionist learning of belief networks. *Artif. Intell.* 56, 71–113. doi: 10.1016/0004-3702(92)90065-6
- Nikovski, D. (2000). Constructing bayesian networks for medical diagnosis from incomplete and partially correct statistics. *IEEE Trans. Knowl. Data Eng.* 4, 509–516. doi: 10.1109/69.868904
- Ostwal, V., and Appenzeller, J. (2019). Spin-orbit torque-controlled magnetic tunnel junction with low thermal stability for tunable random number generation. *IEEE Magn. Lett.* 10, 1–5. doi: 10.1109/LMAG.2019.2912971
- Ostwal, V., Debashis, P., Faria, R., Chen, Z., and Appenzeller, J. (2018). Spin-torque devices with hard axis initialization as stochastic binary neurons. *Sci. Rep.* 8, 1–8. doi: 10.1038/s41598-018-34996-2
- Ostwal, V., Zand, R., DeMara, R., and Appenzeller, J. (2019). A novel compound synapse using probabilistic spin-orbit-torque switching for mtj-based deep neural networks. *IEEE J. Explorat. Solid State Comput. Dev. Circ.* 5, 182–187. doi: 10.1109/JXDCDC.2019.2956468
- Park, D.-C. (2016). Image classification using naive bayes classifier. *Int. J. Comp. Sci. Electron. Eng.* 4, 135–139. Available online at: <http://www.isaet.org/images/extramages/P1216004.pdf>
- Parks, B., Bapna, M., Igbokwe, J., Almasi, H., Wang, W., and Majetich, S. A. (2018). Superparamagnetic perpendicular magnetic tunnel junctions for true random number generators. *AIP Adv.* 8:055903. doi: 10.1063/1.5006422
- Pearl, J. (2014). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier. Available online at: <https://www.elsevier.com/books/probabilistic-reasoning-in-intelligent-systems/pearl/978-0-08-051489-5>
- Pervaiz, A. Z., Ghantasala, L. A., Camsari, K. Y., and Datta, S. (2017). Hardware emulation of stochastic p-bits for invertible logic. *Sci. Rep.* 7:10994. doi: 10.1038/s41598-017-11011-8
- Pervaiz, A. Z., Sutton, B. M., Ghantasala, L. A., and Camsari, K. Y. (2018). Weighted p-bits for fpga implementation of probabilistic circuits. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 1920–1926. doi: 10.1109/TNNLS.2018.2874565
- Premevida, C., Faria, D. R., and Nunes, U. (2017). Dynamic bayesian network for semantic place classification in mobile robotics. *Auton. Robots* 41, 1161–1172. doi: 10.1007/s10514-016-9600-2
- Pufall, M. R., Rippard, W. H., Kaka, S., Russek, S. E., Silva, T. J., Katine, J., et al. (2004). Large-angle, gigahertz-rate random telegraph switching induced by spin-momentum transfer. *Phys. Rev. B* 69:214409. doi: 10.1103/PhysRevB.69.214409
- Querlioz, D., Bichler, O., Vincent, A. F., and Gamrat, C. (2015). Bioinspired programming of memory devices for implementing an inference engine. *Proc. IEEE* 103, 1398–1416. doi: 10.1109/JPROC.2015.2437616
- Rish, I., Brodie, M., Ma, S., Odintsova, N., Beygelzimer, A., Grabarnik, G., et al. (2005). Adaptive diagnosis in distributed systems. *IEEE Trans. Neural Netw.* 16, 1088–1109. doi: 10.1109/TNN.2005.853423
- Roberts, G. O., and Sahu, S. K. (1997). Updating schemes, correlation structure, blocking and parameterization for the gibbs sampler. *J. R. Stat. Soc. Ser. B* 59, 291–317. doi: 10.1111/1467-9868.00070
- Russell, S. J., and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Pearson Education Limited. Available online at: <https://www.pearson.com/us/higher-education/product/Norvig-Artificial-Intelligence-A-Modern-Approach-Subscription-3rd-Edition/9780133001983.html>
- Shim, Y., Chen, S., Sengupta, A., and Roy, K. (2017). Stochastic spin-orbit torque devices as elements for bayesian inference. *Sci. Rep.* 7:14101. doi: 10.1038/s41598-017-14240-z
- Strukov, D., Indiveri, G., Grollier, J., and Fusi, S. (2019). Building brain-inspired computing. *Nat. Commun.* 10. doi: 10.1038/s41467-019-12521-x
- Sun, S., Zhang, C., and Yu, G. (2006). A bayesian network approach to traffic flow forecasting. *IEEE Trans. Intell. Transport. Syst.* 7, 124–132. doi: 10.1109/TITS.2006.869623
- 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., Jaiswal, R., Camsari, K. Y., and Datta, S. (2020). Autonomous probabilistic coprocessing with petaflops per second. *IEEE Access*. 157238–157252. doi: 10.1109/ACCESS.2020.3018682
- Thakur, C. S., Afshar, S., Wang, R. M., Hamilton, T. J., Tapson, J., and Van Schaik, A. (2016). Bayesian estimation and inference using stochastic electronics. *Front. Neurosci.* 10:104. doi: 10.3389/fnins.2016.00104
- Ticknor, J. L. (2013). A bayesian regularized artificial neural network for stock market forecasting. *Expert Syst. Appl.* 40, 5501–5506. doi: 10.1016/j.eswa.2013.04.013
- Torunbalci, M. M., Upadhyaya, P., Bhawe, S. A., and Camsari, K. Y. (2018). Modular compact modeling of MTJ devices. *IEEE Trans. Electron Dev.* 65, 4628–4634. doi: 10.1109/TED.2018.2863538
- Tyelman, W., Waszyrowski, T., Napieralski, A., Kamiński, M., Trafidlo, T., Kulesza, Z., et al. (2016). Real-time prediction of acute cardiovascular events using hardware-implemented bayesian networks. *Comput. Biol. Med.* 69, 245–253. doi: 10.1016/j.compbiomed.2015.08.015
- Vodenicarevic, D., Locatelli, N., Mizrahi, A., Friedman, J. S., Vincent, A. F., Romera, M., et al. (2017). Low-energy truly random number generation with superparamagnetic tunnel junctions for unconventional computing. *Phys. Rev. Appl.* 8:054045. doi: 10.1103/PhysRevApplied.8.054045
- Vodenicarevic, D., Locatelli, N., Mizrahi, A., Hirtzlin, T., Friedman, J. S., Grollier, J., et al. (2018). “Circuit-level evaluation of the generation of truly random bits with superparamagnetic tunnel junctions,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (Florence: IEEE), 1–4. doi: 10.1109/ISCAS.2018.8351771
- Weijia, Z., Ling, G. W., and Seng, Y. K. (2007). “PCMOs-based hardware implementation of bayesian network,” in *2007 IEEE Conference on Electron Devices and Solid-State Circuits* (Tainan: IEEE), 337–340. doi: 10.1109/EDSSC.2007.4450131
- Zand, R., Camsari, K. Y., Pyle, S. D., Ahmed, I., Kim, C. H., and DeMara, R. F. (2018). “Low-energy deep belief networks using intrinsic sigmoidal spintronic-based probabilistic neurons,” in *Proceedings of the 2018 on Great Lakes Symposium on VLSI* Chicago IL, 15–20. doi: 10.1145/3194554.3194558
- Zermani, S., Dezan, C., Chenini, H., Diguët, J.-P., and Euler, R. (2015). “FPGA implementation of bayesian network inference for an embedded diagnosis,” in *2015 IEEE Conference on Prognostics and Health Management (PHM)* (Austin, TX: IEEE), 1–10. doi: 10.1109/ICPHM.2015.7245057
- Zink, B. R., Lv, Y., and Wang, J.-P. (2018). Telegraphic switching signals by magnet tunnel junctions for neural spiking signals with high information capacity. *J. Appl. Phys.* 124:152121. doi: 10.1063/1.5042444
- Zou, M., and Conzen, S. D. (2004). A new dynamic bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics* 21, 71–79. doi: 10.1093/bioinformatics/bth463

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 Faria, Kaiser, 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.



Oscillatory Neural Networks Using VO₂ Based Phase Encoded Logic

Juan Núñez^{1*}, María J. Avedillo¹, Manuel Jiménez¹, José M. Quintana¹,
Aida Todri-Sanial², Elisabetta Corti³, Siegfried Karg³ and Bernabé Linares-Barranco¹

¹ Instituto de Microelectrónica de Sevilla (IMSE-CNM), CSIC and Universidad de Sevilla, Seville, Spain, ² Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier, Montpellier, France,

³ Department of Science and Technology, IBM Research – Zurich, Rüschlikon, Switzerland

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Robert W. Newcomb,
University of Maryland, College Park,
United States
Mauro Forti,
University of Siena, Italy

*Correspondence:

Juan Núñez
jnunez@imse-cnm.csic.es

Specialty section:

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

Received: 19 January 2021

Accepted: 25 March 2021

Published: 16 April 2021

Citation:

Núñez J, Avedillo MJ, Jiménez M,
Quintana JM, Todri-Sanial A, Corti E,
Karg S and Linares-Barranco B
(2021) Oscillatory Neural Networks
Using VO₂ Based Phase Encoded
Logic. *Front. Neurosci.* 15:655823.
doi: 10.3389/fnins.2021.655823

Nano-oscillators based on phase-transition materials are being explored for the implementation of different non-conventional computing paradigms. In particular, vanadium dioxide (VO₂) devices are used to design autonomous non-linear oscillators from which oscillatory neural networks (ONNs) can be developed. In this work, we propose a new architecture for ONNs in which sub-harmonic injection locking (SHIL) is exploited to ensure that the phase information encoded in each neuron can only take two values. In this sense, the implementation of ONNs from neurons that inherently encode information with two-phase values has advantages in terms of robustness and tolerance to variability present in VO₂ devices. Unlike conventional interconnection schemes, in which the sign of the weights is coded in the value of the resistances, in our proposal the negative (positive) weights are coded using static inverting (non-inverting) logic at the output of the oscillator. The operation of the proposed architecture is shown for pattern recognition applications.

Keywords: phase transition materials, VO₂, nano-oscillators, ONNs, neuromorphics

INTRODUCTION

Phase-transition materials (PTMs) like vanadium dioxide (VO₂), with their abrupt switching between states with very different resistivity, are being explored for implementing non-boolean computational paradigms such as neuromorphic architectures. In particular, different groups are exploiting the capability of a PTM device in series with a resistor to oscillate in the implementation of oscillator based computing (OBC).

The field of OBC is not a new idea, with outstanding contributions in the field of logic in the 1950s (von Neumann, 1957; Goto, 1959). In recent years, this idea has received considerable interest and has become an active research area due to the appearance of devices, operating based on very different physical phenomena, with the ability to implement very compact oscillators and with very low energy consumption.

In Csaba and Porod (2020) numerous oscillators are evaluated as potential building blocks of OBC. In terms of energy, PTM-based relaxation oscillators show good performance. They are reported to reduce energy per cycle by more than an order of magnitude when compared to CMOS ring oscillators (ROs). They rank second in terms of energy efficiency, behind only superconducting oscillators.

The most widely used compound as phase-transition material is VO₂ and the term VO₂ nano-oscillator has come to be coined (Maffezzoni et al., 2015; Sharma et al., 2015). In addition, they

show good performance in terms of scalability and interconnection with electronic circuits, without requiring any conversion between electrical variables and other non-electrical variables as occurs with other nano-oscillators that exploit other physical magnitudes. Numerous experimental results of VO₂ nano-oscillators have been reported as well as some preliminary results of applications (Shukla et al., 2016; Corti et al., 2018, 2020; Dutta et al., 2019a,b).

Oscillator based computing encompasses a wide variety of operating principles and architectures. In the first place, one can distinguish between those that work with oscillators of ideally identical frequency and the processing corresponds to obtaining a pattern of phase synchronization, phase shift key (PSK) and those that work with oscillators of different frequencies and patterns of frequency synchronization frequency shift key (FSK) (Nikonov et al., 2015).

Oscillator-based-computing phase-shift-key has been applied to obtain solutions to combinatorial optimization problems, difficult to solve in conventional computers. In Wu et al. (2011); Parihar et al. (2017), the problem of graph coloring is solved from the steady-state of a network of oscillators, which represent the nodes, and in which the branches of the corresponding graph are mapped into interconnections that push to separate the phases of the adjacent oscillators. In Dutta et al. (2019a), the resolution of a Max-Cut problem using VO₂ oscillators is shown experimentally. The comparison with other implementations in terms of scalability, power and quality of the results obtained is very favorable.

Phase shift key has been also applied to explore oscillatory neural networks (ONNs) using a Hopfield-type architecture for associative memories with application in pattern recognition. The neurons in the network are replaced by oscillators and the output is determined by the phase of each one. There are contributions of mathematical analysis with simulations using phase models for neurons (Hoppensteadt and Izhikevich, 1999; Follmann et al., 2015) as well as reporting implementations with different types of oscillators [phase-locked loops and voltage-controlled oscillators (Hoppensteadt and Izhikevich, 2000), non-volatile logic based on magnetic tunnel junctions (Calayir and Pileggi, 2013), micro-electro-mechanical systems and a feedback loop with transconductance amplifiers (Kumar and Mohanty, 2017), comparator and a digital circuit in Jackson et al. (2018), CMOS ring oscillators (Csaba et al., 2016), STOs (Popescu et al., 2018), or VO₂ (Shukla et al., 2014; Maffezzoni et al., 2015; Corti et al., 2018)]. The implementations based on VO₂ devices exhibit potential for very low energy computation (Corti et al., 2020). In the case of electrical oscillators, synapses are implemented with resistors or memristors that play the role of weights. In this way, the output of each neuron interacts electrically with the rest. Recently, the potential of ONNs with a small number of neurons to efficiently tackle different image processing tasks has been revealed. Corti et al. (2021) have shown that this approach using VO₂ oscillators can be exploited for the implementation of commercial high-accuracy image processing architectures based on convolutional neural networks (CNN).

Motivated by the latter type of application, in this paper we describe the implementation of an ONN using VO₂ based phase

encoded logic (PeL). PeL with VO₂ devices has been recently proposed (Avedillo et al., 2020) by the authors. It uses the phase to encode information in logic circuits and its basic building block is a VO₂ oscillator which performs a weighted sum of inputs to evaluate its output phase. So we propose to use it to build an ONN. It overcomes some limitations of previously reported VO₂ ONNs (Corti et al., 2018, 2020).

MATERIALS AND METHODS

Background

ONNs With VO₂ Oscillators

Figure 1 shows the ONN proposed in Corti et al. (2018, 2020), and the VO₂ oscillator used as neuron. The resistances implement the synapses among neurons.

Under no electrical stimuli VO₂ tends to stabilize in the insulating state. When the applied voltage increases and so the current density that flows through it reaches a critical current density, J_{C-MIT} , insulator to metal transition (IMT) occurs. When the voltage decreases and so the current density reduces below J_{C-MIT} , the metal to insulator transition (MIT) takes place, transitioning from the metallic to the insulating state. Electrical parameters of its model, are summarized in Table 1. V_{IMT} and V_{MIT} are the voltages at which the IMT and MIT transition occur, respectively. R_{INS} and R_{MET} are the resistances in the insulating and metallic state. Since MIT and IMT transitions are abrupt but not instantaneous, transition times (TT_{IMT} and TT_{MIT}) are also included. The I-V characteristic of such device is depicted in Figure 2A. The VO₂ device has been simulated with a behavioral model as described in Maffezzoni et al. (2015). As expected, in the insulating operating zone the slope is significantly flat, which indicates that the resistance value is very high. On the other hand, in the metallic state, the slope of the I-V curve is clearly steeper, thus implying that the resistance is lower.

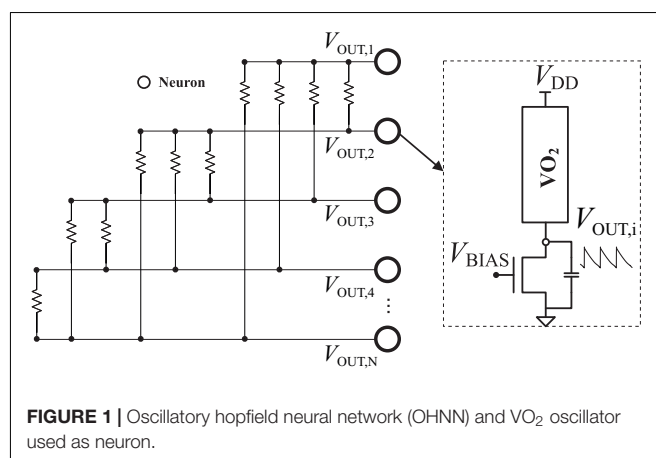


TABLE 1 | Vanadium dioxide (VO₂) electrical parameters.

V_{IMT}	V_{MIT}	R_{MET}	R_{INS}	TT
1.99 V	0.99 V	0.99 K Ω	100.2 K Ω	30 ns

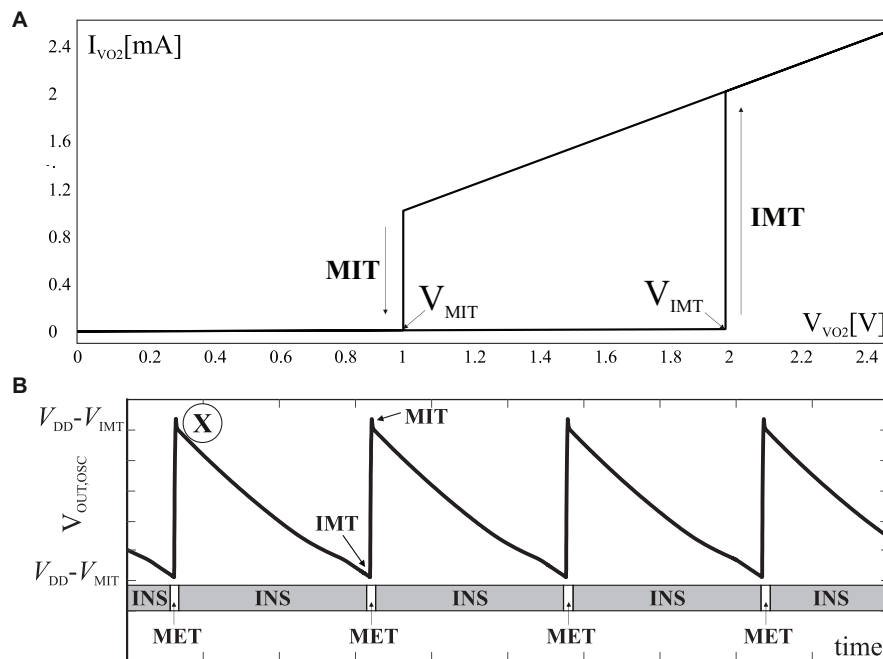


FIGURE 2 | (A) I-V curve of the VO₂ device reported in Corti et al. (2018, 2020). **(B)** Simulated waveform of the output of a VO₂ oscillator.

Figure 2B depicts simulated waveforms for the oscillator output, $V_{OUT,OSC}$, with $V_{DD} = 2.5$ V and $V_{BIAS} = 2.5$ V. The state of the VO₂ device is also shown to better illustrate the circuit behavior. Regions marked with “INS” label mean that the device is in the insulating state, whereas those marked with “MET” corresponds to the device in the metallic state. When the VO₂ is in an insulating state (point “X” in **Figure 2B**), the oscillator output is discharged through the transistor and, therefore, the voltage drop across the VO₂ ($V_{DD} - V_{OUT,OSC}$) and the current through this device are increased. When the circulating current density reaches the critical value J_{C-IMT} , the VO₂ switches to the metallic state. On the other hand, the switching to the metallic state occurs once the VO₂ voltage reaches V_{IMT} , when the output is then charged through the VO₂ device. Due to the low R_{MET} value, this charging is very fast and leads to a reduction of the voltage seen by the VO₂ until it reaches V_{MIT} and the MIT occurs. Finally, note that the voltage V_{BIAS} can be used to control the frequency of the signal.

This ONN works as an associative memory with application in pattern recognition. The ONN state is defined by the phase of each neuron. There are states which are stable and others which if entered converge to a stable one. For pattern recognition, a set of patterns (called training patterns) are said to be stored in the network, which means the network is configured so that the state corresponding to such patterns are stable. When the network is placed in a state corresponding to a distorted version of a training pattern, it evolves to a training pattern, ideally to the most similar one. Placing the ONN in a given state means fixing a specific phase for each oscillator. This is achieved by suitably delaying the switching on of the supply voltage of each oscillator.

The stable states of the network are determined by the resistance values connecting the oscillators, which plays the role of the weights of the neural network. The required weights to store a given set of training patterns are derived applying the well-known Hebbian rule (Hebb, 1949) and then mapped to resistance values.

There are several challenges in the operation of this ONN. First, in order to work properly, the neurons must all synchronize in frequency. Although ideally all neurons are identical, and so they oscillate at the same frequency, in practice this is not the case because of different reasons. Variability between the VO₂ devices is the main one. Secondly, both positive and negative values must be mapped to resistance values. Note that positive weights mean that the phase of both associated oscillators should be pulled to each other while negative ones should have the contrary effect. Although, there are results showing that two oscillators coupled with enough large resistance value end in anti-phase configuration, the device-to-device of variability can have a great impact on this behavior, especially when many oscillators are coupled. The ONN we propose aims at addressing these challenges.

VO₂-Based PeL Description

Key components of PeL logic are VO₂ oscillators with only two possible phases, 180° apart. The oscillating phase depends on the phases (also discretized) of the applied inputs. That is, the resulting phase is a logic function of the inputs. In particular, it implements a majority functionality. The output phase is the majority phase.

Figure 3A depicts the schematic of a three-input majority gate (Avedillo et al., 2020). It exploits sub harmonic injection locking

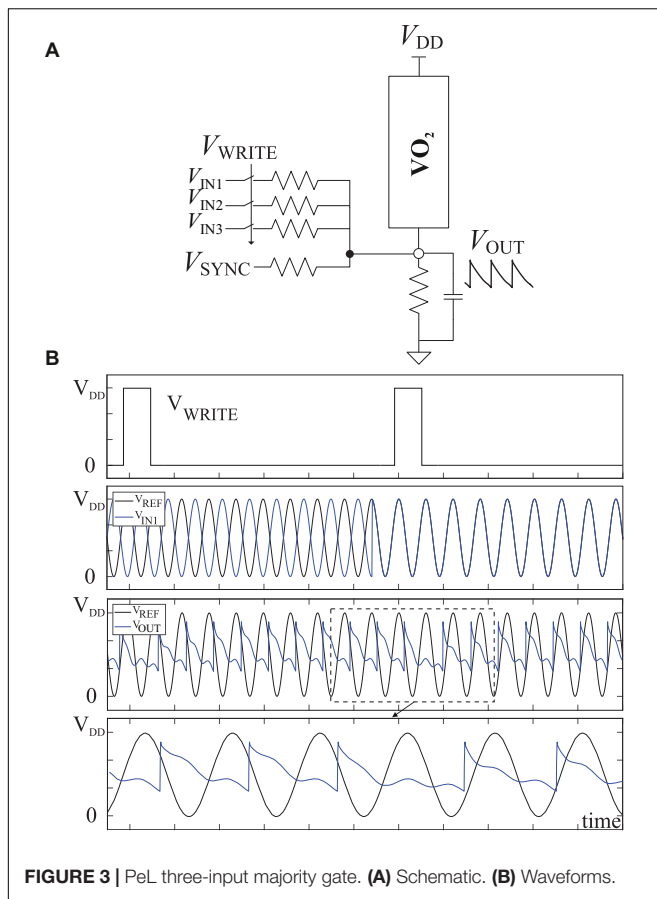


FIGURE 3 | PeL three-input majority gate. **(A)** Schematic. **(B)** Waveforms.

(SHIL) to stabilize the oscillator frequency against variability effects and to discretize its phase. This is achieved by injecting a synchronization signal (V_{SYNC}), which ranges between 0 and V_{DD} , of frequency close to twice the natural frequency of the oscillator. The oscillator outputs exhibit half such injected frequency. The phases of the input signal are represented in terms of that of a reference signal (V_{REF}) external to the oscillator.

When signal V_{WRITE} activates, the oscillator phase is forced to that of the majority phase of its three inputs. For example, for $(V_{\text{IN1}}, V_{\text{IN2}}, V_{\text{IN3}}) = (\overline{V_{\text{REF}}}, \overline{V_{\text{REF}}}, V_{\text{REF}})$, the phase corresponding to $\overline{V_{\text{REF}}}$ is stored and for $(V_{\text{IN1}}, V_{\text{IN2}}, V_{\text{IN3}}) = (V_{\text{REF}}, \overline{V_{\text{REF}}}, V_{\text{REF}})$ the phase corresponding to V_{REF} is stored. **Figure 3B** depicts simulation results for those input combinations. From top to bottom the V_{WRITE} , V_{IN1} and the oscillator output, V_{OUT} , are shown. V_{IN1} is the only changing input and determines the output value. A reference signal (V_{REF}) is also displayed to ease identification of the phase of each signal. That is, V_{IN1} is $\overline{V_{\text{REF}}}$ initially and then changes to V_{REF} . Note the phase change of V_{OUT} in response to the application of the second V_{WRITE} pulse.

PeL-Based ONN Architecture

The New Neuron

Figure 4A depicts the topology proposed for the neuron. Note the use of the synchronization signal V_{SYNC} , like in PeL, although

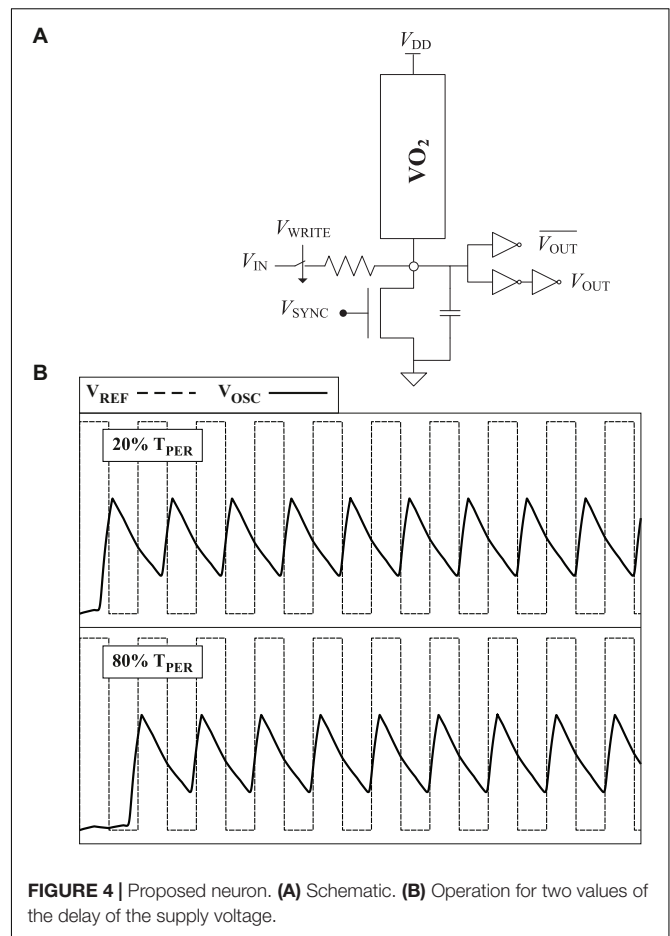


FIGURE 4 | Proposed neuron. **(A)** Schematic. **(B)** Operation for two values of the delay of the supply voltage.

it is injected through the gate of the transistor avoiding the injection resistor. In the neuron, it contributes to reducing period variations due to device variability, which translates to frequency synchronization advantages in the ONN network. **Figure 4B** depicts its operation. The oscillator output before the static logic, V_{OSC} , and a reference signal are shown. The use of SHIL reduces the number of phases to two. It can be observed that the oscillation phase can be controlled by the supply voltage delay like in the original ONN described in the background section. Supply voltages delays under half the oscillation period (top waveform in **Figure 4B**) lead to one phase and delays over half period (bottom waveform) force the other phase.

It is interesting to study the robustness of the ONNs against variations in the electrical parameters of the VO₂ (resistance in the insulating and metallic state and switching voltages between states). In this sense, conventional implementations of ONNs are sensitive to these variations. **Figure 5A** depicts a design space for the phase difference between two identical coupled oscillators in which the variables are the time difference between oscillator initialization (ΔT) and the coupling resistance (R_C). Note how two clearly differentiated regions corresponding to an in-phase (0°) and out-of-phase (180°) operation are observed. **Figure 5B** reproduces the previous plot by considering variations of 10% of the insulating and metallic resistances of

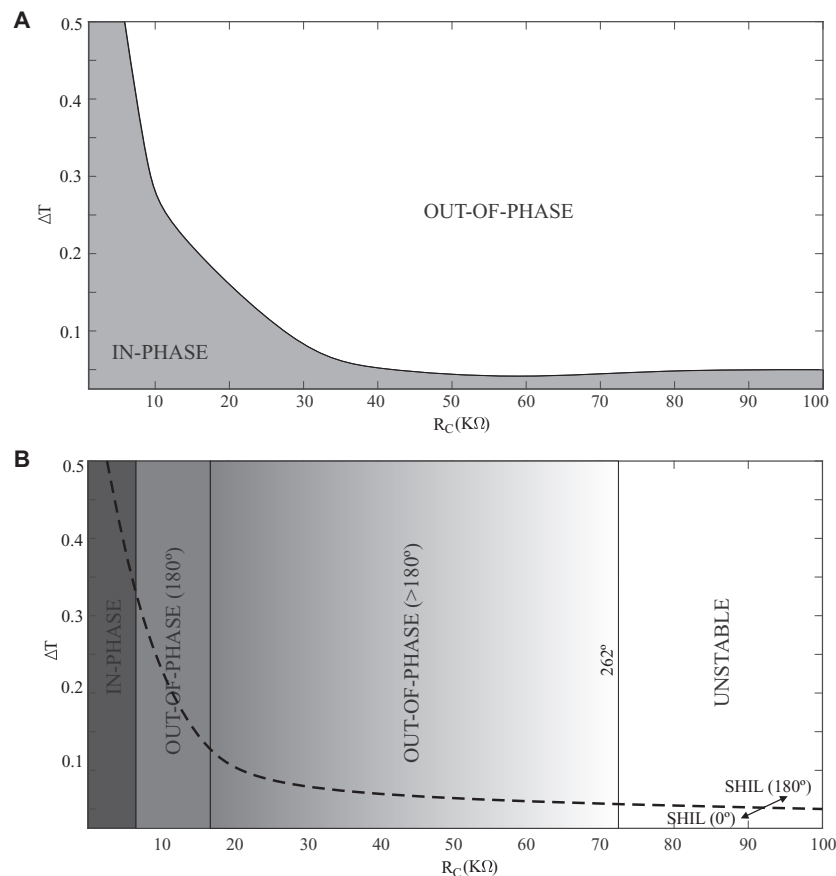


FIGURE 5 | ΔT vs. R_C plot for in-phase/out-of-phase operation of two conventional coupled oscillators. **(A)** Nominal scenario. **(B)** Considering R_{INS} and R_{MET} variations. The boundary for the in-phase/out-of-phase operation of two SHIL-based oscillators is depicted with a dashed line.

the VO₂ device of one of the oscillators with respect to the nominal scenario ($R_{INS,1} = 0.9 \cdot R_{INS,2}$ and $R_{MET,1} = 0.9 \cdot R_{MET,2}$). Significant differences are observed given that there are phase differences other than 180° for the out-of-phase region and even an area of unstable behavior. Unstable behavior means that both oscillators are not able to synchronize. Also, note that in the plot corresponding to ideal oscillators there are resistance values for which both in-phase and out-of-phase are possible depending on the initial delay (phase difference between) the two oscillators. This bistability, which is interesting from the point of view of the ONN functionality, is not observed in the plot with variability (**Figure 5B**). In this figure, the boundary between the in-phase and out-of-phase operating regions for two coupled oscillators using SHIL, and considering the same variation between the VO₂ resistances, has been represented using a dashed line. Two clearly distinct regions of operation are observed like in the ideal plot. These results reveal that SHIL has significant benefits in that inherently two complementary phases are obtained at the output and this is more tolerant to variations in the parameters of VO₂.

Synapse

The proposed interconnection scheme encodes the sign of the weights in the way the neurons are connected unlike the original

ONN, which relies just on resistance values. **Figure 6A** shows two possible scenarios for the interconnection of two neurons using positive and negative weights. When interconnecting using positive weights the output of each is connected to a buffer (marked in green), while for encoding negative weights an inverter (marked in red) is used. Note that unlike the original ONN, which uses a bidirectional interconnection mechanism with a single coupling resistor between two neurons, in the proposed scheme the interconnection is unidirectional and therefore two coupling resistors must be used. The rationale behind using the oscillation signal but complemented for negative weights is that in an ONN, a negative weight must push away the phases of the two neurons which is equivalent to pull the phase of the neurons to the complement of the other one. The buffer is used for positive weights so that the shape of the two outputs of the neuron are similar. **Figure 6** also shows simulation results of both interconnection schemes. The four scenarios are illustrated. Initially in-phase neurons connected with positive coupling weight (**Figure 6B**) and with negative weight (**Figure 6C**) and initially out-of-phase neurons connecting with positive (**Figure 6D**) and negative (**Figure 6E**) weights. Note neurons coupled with positive weights end up in phase independently of their initial states (**Figures 6B,D**).

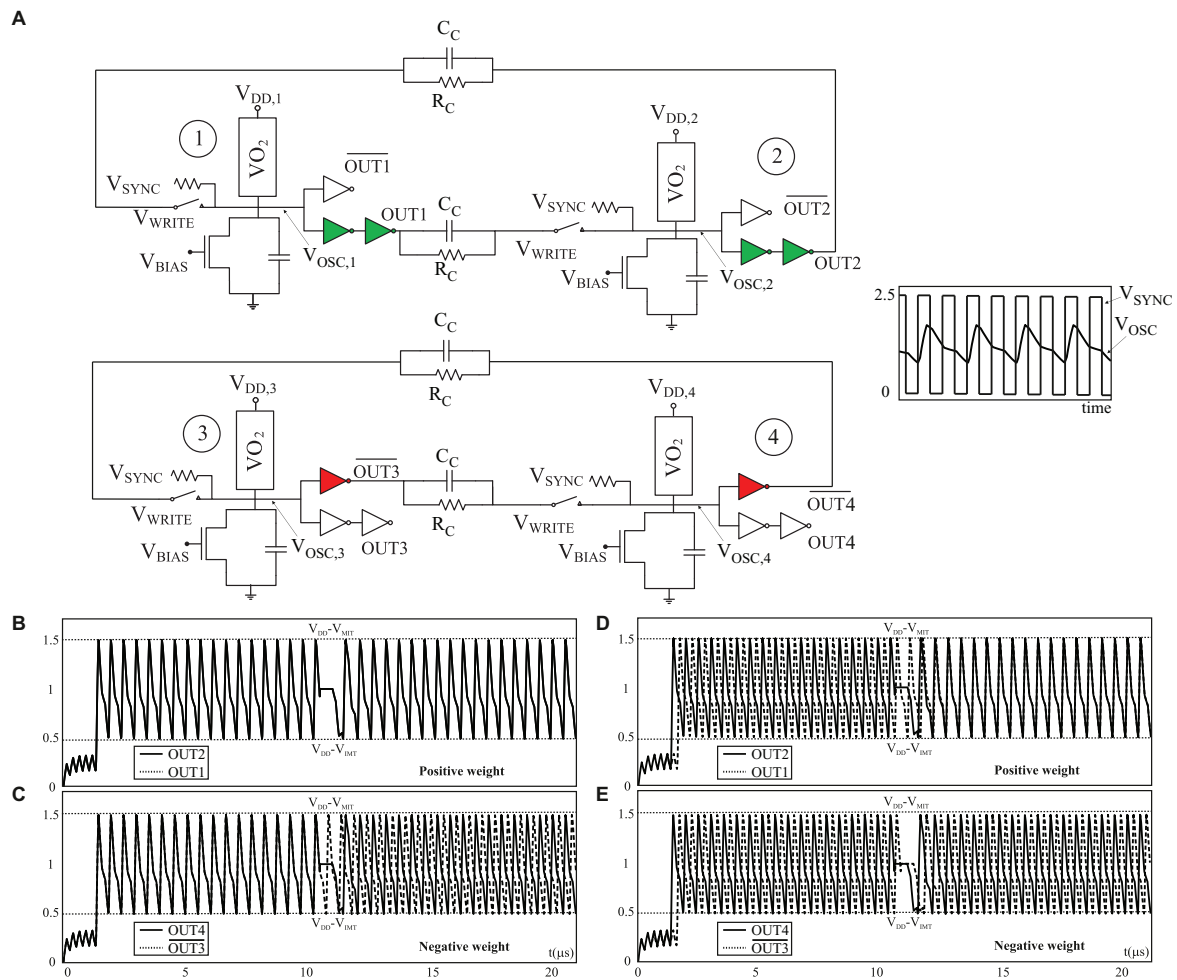


FIGURE 6 | (A) Schematics of the interconnection of two neurons with a positive (green) and a negative (red) weight. **(B–E)** Waveforms showing the operation of both interconnection schemes.

Similarly, neurons coupled with negative weights evolve toward out of phase (Figures 6C,E).

Network Operation

The switch at the input of the neurons allows disconnecting the coupling among them by fixing the V_{WRITE} signal to a low voltage. This is used at the beginning of the operation to initialize the ONN state (phase of each neuron oscillation). As mentioned before, this is the way an input pattern is applied to the network. After application of successively positive V_{WRITE} pulses enable interaction among the neurons and the network state evolves toward ideally the closest stored pattern.

RESULTS

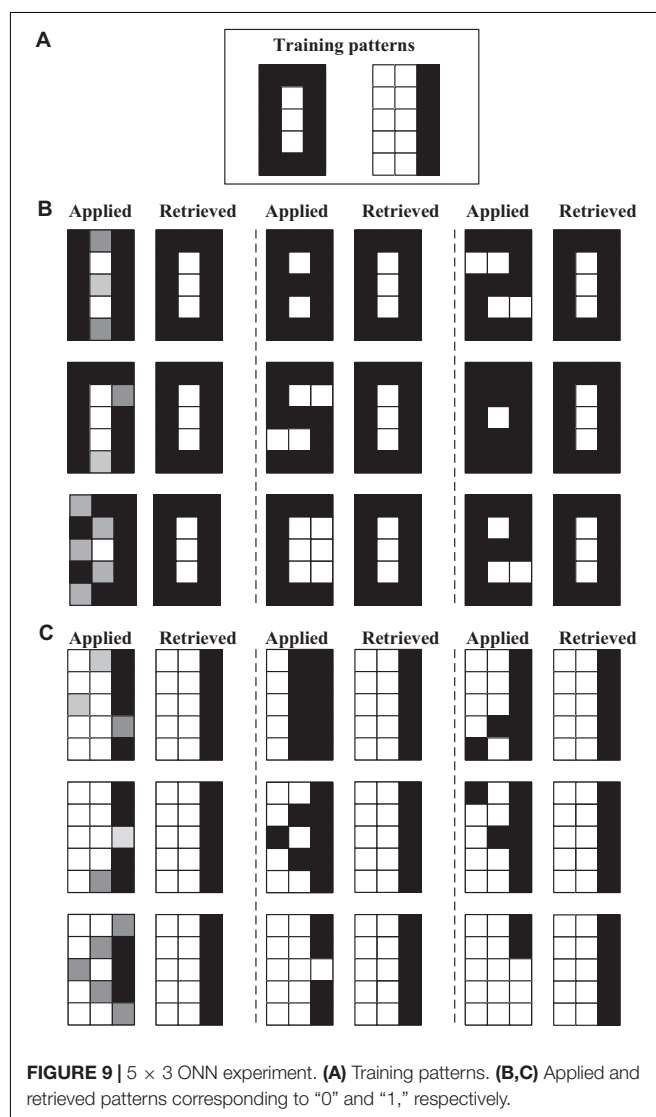
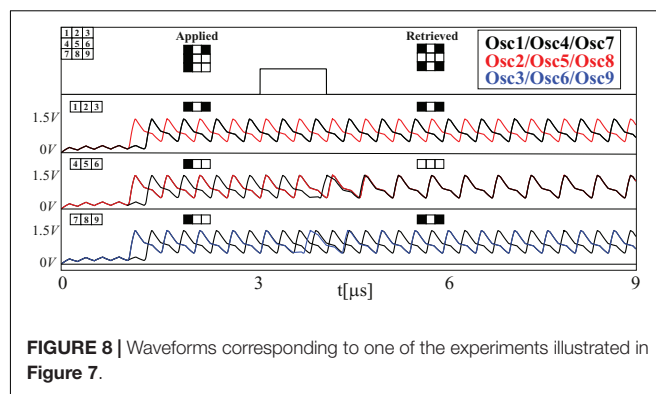
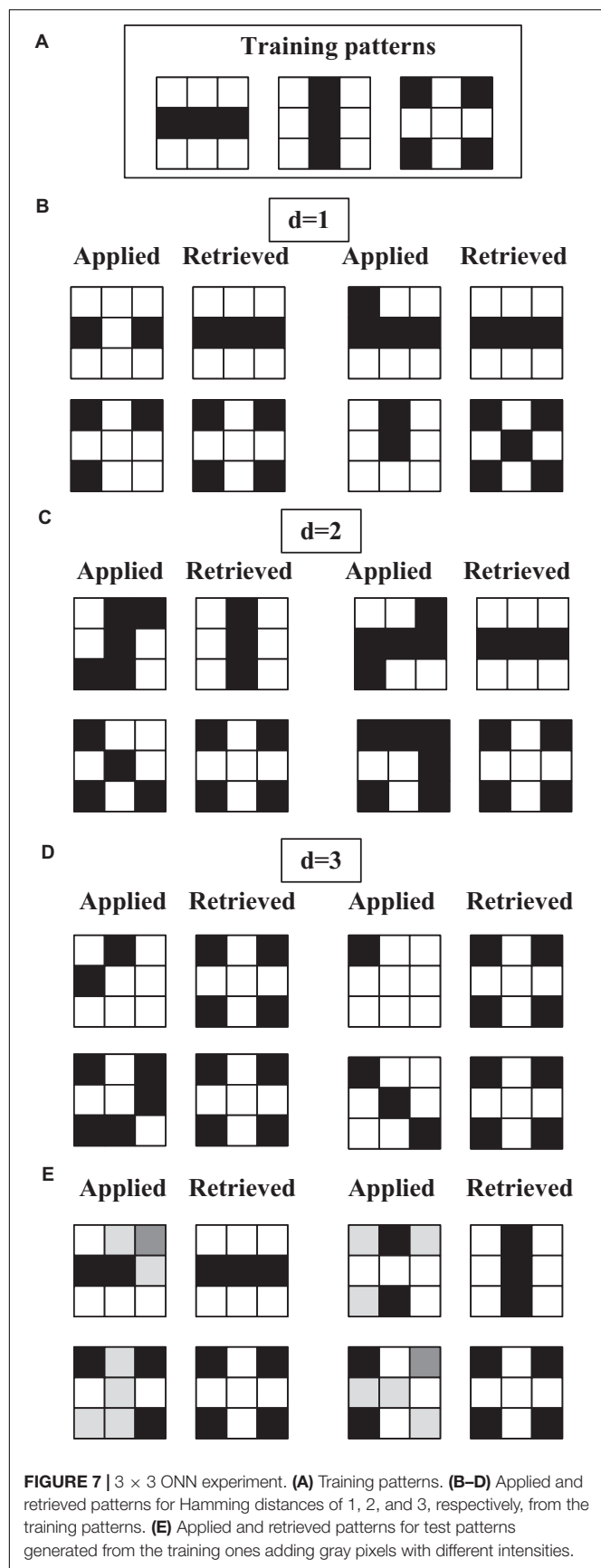
PeL Associative Memory

As a first experiment to verify the operation of the network, we propose the training and test patterns corresponding to 3×3 pixel size images shown in Figure 7. In this demonstration (and

the following ones), the supply voltage is 2.5 V, the oscillator capacitance is 100 pF and the coupling capacitance is 0.05 pF. The obtained Hebbian weight matrix exhibits two positive and two negative values which have been mapped to resistances, 100 K Ω and 300 K Ω , respectively.

The results are shown in Figures 7B–E. Waveforms corresponding to one of these experiments are shown in Figure 8. Specifically, the write pulse and the outputs of the oscillators before the inverter/buffer are shown. Note that the outputs have been grouped for each of the rows. After the first writing cycle, the outputs of bits 4 and 9 change their phase and, thus, the expected training pattern is recovered.

Coming back to Figure 7, the test patterns for which results are shown have been categorized into four groups in order to facilitate the analysis. The first three groups represent test patterns at Hamming distances of 1 (Figure 7B), 2 (Figure 7C), and 3 (Figure 7D) from the training or stored patterns. To consider a result as correct, the retrieved pattern must match the training pattern that has the lowest Hamming distance with respect to the test pattern. All but



one test pattern are correctly retrieved. In fact, the one that has not been recovered, the vertical line with a missed pixel in **Figure 7B**, is at distance of 1. However, in order to fairly analyze the pattern recognition performance of the proposed ONN, it is important to be aware of the capabilities

of the Hopfield model itself. It is well known that even the ideal model is not able to correctly retrieve any number of stored patterns, but its capacity depends on the number of neurons, the correlation among the patterns to be stored, and the learning rule. In the case of random training patterns, the maximum number which can be reliably stored ($P_{\text{ERROR}} < 1/N$) is $0.14 \cdot N$ for the Hebbian learning rule (Hopfield, 1982). That is, we should not expect perfect retrieval since we are storing too many patterns for the network size. So, it is interesting to investigate also the performance of the Hopfield network on this example. For that, the same example has been simulated with a MATLAB model of a Hopfield network. Its accuracy is under 80.5% and in particular, neither it recovers the vertical line from the one without the bottom pixel. From our simulations, we have also observed that the third training pattern is easier to be retrieved and we have confirmed that this is also the case for the model. This is in agreement with this pattern being the one exhibiting the smaller energy minimum and so exercising stronger attraction.

The last group (Figure 7E) depicts test patterns generated from the training ones adding gray pixels with different intensities. As explained before, gray values in the input image are encoded in distinct initialization times of the oscillators. It can be observed that the most similar training pattern is retrieved in all cases.

PeL ONN for Character Recognition

In order to further illustrate our proposal, we have designed an ONN for character recognition. For this, using the Hebbian rule, weights have been derived to store 5×3 pixels representations of digits “0” and “1” as shown in Figure 9A. These weights have been mapped to resistance values ($R_C = 200 \text{ K}\Omega$ and $R_C = 400 \text{ K}\Omega$ for strong and weak coupling weights, respectively) and an ONN with our architecture has been simulated with them using HSpice electrical simulator.

The performance of ONN has been evaluated using the set of 18 test patterns shown in Figure 9. These experiments have been grouped in sets of nine corresponding to an expected output of the image “0” (Figure 9B) and “1” (Figure 9C) based on the criterion of minimum Hamming distance, in which both the applied and retrieved patterns are shown. Within each set of 9, the first column represents noisy versions of the corresponding training pattern. The other two columns represent harder test

patterns in which some bits have been completely flipped. The results show that all applied patterns were successfully retrieved.

DISCUSSION

A novel ONN architecture based on phase encoding is proposed and its operation as associative memory is shown. Phase information storage using oscillators with VO₂ devices and subharmonic injection locking is exploited for the neurons. SHIL has shown to greatly increase the variability robustness with respect to free VO₂ oscillators. The proposed mechanism of interconnecting neurons encodes the sign of the weight by using static logic to force a phase change instead of just having different resistance values. This architecture allows overcoming some of the challenges that arise in other implementations of VO₂-based ONN, including improved robustness against variability and simplifying mapping of weights to resistance values.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

JN, MA, and JQ developed the main concepts. JN performed all the simulations. All authors assisted in the writing of the manuscript and developing the concepts.

FUNDING

This work has been funded by NeurONN Project (Horizon 2020 – Grant agreement ID: 871501) and Ministerio de Economía y Competitividad del Gobierno de España with support from FEDER (Project TEC2017-87052-P). We acknowledge support of the publication fee by the CSIC Open Access Publication Support Initiative through its Unit of Information Resources for Research (URICI).

REFERENCES

- Avedillo, M. J., Quintana, J. M., and Núñez, J. (2020). Phase transition device for phase storing. *IEEE Trans. Nanotechnol.* 19, 107–112. doi: 10.1109/TNANO.2020.2965243
- Calayir, V., and Pileggi, L. (2013). “Fully-digital oscillatory associative memories enabled by non-volatile logic,” in *Proceedings of the 2013 International Joint Conference on Neural Networks*, Dallas, TX. doi: 10.1109/IJCNN.2013.6706925
- Corti, E., Cornejo, J. A., Niang, K. M., Robertson, J., Moselund, K. E., Gotsmann, B., et al. (2021). Coupled VO₂ oscillators circuit as analog first layer filter in convolutional neural networks. *Front. Neurosci.* 15:628254. doi: 10.3389/fnins.2021.628254
- Corti, E., Gotsmann, B., Moselund, K., Stolichnov, I., Ionescu, A., and Karg, S. (2018). “Resistive coupled VO₂ oscillators for image recognition,” in *Proceedings of the 2018 IEEE International Conference on Rebooting Computing*, McLean, VA. doi: 10.1109/ICRC.2018.8638626
- Corti, E., Khanna, A., Niang, K., Robertson, J., Moselund, K., Gotsmann, B., et al. (2020). Time-delay encoded image recognition in a network of resistively coupled VO₂ on Si oscillators. *IEEE Electron Device Lett.* 41, 629–632. doi: 10.1109/LED.2020.2972006
- Csaba, G., and Porod, W. (2020). Coupled oscillators for computing: a review and perspective. *Appl. Phys. Rev.* 7:011302. doi: 10.1063/1.5120412
- Csaba, G., Ytterdal, T., and Porod, W. (2016). “Neural network based on parametrically-pumped oscillators,” in *Proceedings of the IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Monte Carlo, 45–48. doi: 10.1109/ICECS.2016.7841128
- Dutta, S., Khanna, A., Gomez, J., Ni, K., Toroczka, Z., and Datta, S. (2019a). “Experimental demonstration of phase transition nano-oscillator based Ising machine,” in *Proceedings of the IEEE International Electron Devices Meeting*

- (IEDM), San Francisco, CA, 37.8.1–37.8.4. doi: 10.1109/IEDM19573.2019.8993460
- Dutta, S., Parihar, A., Khanna, A., and Gomez, J. (2019b). Programmable coupled oscillators for synchronized locomotion. *Nat. Commun.* 10:3299. doi: 10.1038/s41467-019-11198-6
- Follmann, R., Macau, E. E., Rosa, E. Jr., and Piqueira, J. R. (2015). Phase oscillatory network and visual pattern recognition. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1539–1544. doi: 10.1109/TNNLS.2014.2345572
- Goto, E. (1959). The parametron, a digital computing element which utilizes parametric oscillation. *Proc. IRE* 47, 1304–1316.
- Hebb, D. O. (1949). *The Organization of Behavior*. New York, NY: Wiley.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational capabilities. *Proc. Natl. Acad. Sci. U.S.A.* 79, 2554–2558.
- Hoppensteadt, F. C., and Izhikevich, E. M. (1999). Oscillatory neurocomputers with dynamic connectivity. *Phys. Rev. Lett.* 82:2983. doi: 10.1103/PhysRevLett.82.2983
- Hoppensteadt, F. C., and Izhikevich, E. M. (2000). Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE Trans. Neural Netw.* 11, 734–738. doi: 10.1109/72.846744
- Jackson, T., Pagliarini, S., and Pileggi, L. (2018). “An oscillatory neural network with programmable resistive synapses in 28 nm CMOS,” in *Proceedings of the 2018 IEEE International Conference on Rebooting Computing (ICRC)*, McLean, VA, 1–7. doi: 10.1109/ICRC.2018.8638600
- Kumar, A., and Mohanty, P. (2017). Autoassociative memory and pattern recognition in micromechanical oscillator network. *Sci. Rep.* 7:411. doi: 10.1038/s41598-017-00442-y
- Maffezzoni, P., Daniel, L., Shukla, N., Datta, S., and Raychowdhury, A. (2015). Modeling and simulation of vanadium dioxide relaxation oscillators. *IEEE Trans. Circuits Syst. I Regul. Pap.* 62, 2207–2215. doi: 10.1109/TCSI.2015.2452332
- Nikonov, D. E., Csaba, G., Porod, W., Shibata, T., Voils, D., Hammerstrom, D., et al. (2015). Coupled-oscillator associative memory array operation for pattern recognition. *IEEE J. Explor. Solid State Comput. Devices Circuits* 1, 85–93. doi: 10.1109/XCDC.2015.2504049
- Parihar, A., Shukla, N., Jerry, M., Datta, S., and Raychowdhury, A. (2017). Vertex coloring of graphs via phase dynamics of coupled oscillatory networks. *Sci. Rep.* 7:911. doi: 10.1038/s41598-017-00825-1
- Popescu, B., Csaba, G., Popescu, D., Fallahpour, A. H., Lugli, P., Porod, W., et al. (2018). Simulation of coupled spin torque oscillators for pattern recognition. *J. Appl. Phys.* 124:152128. doi: 10.1063/1.5042423
- Sharma, A., Bain, J. A., and Weldon, J. A. (2015). Phase coupling and control of oxide-based oscillators for neuromorphic computing. *IEEE J. Explor. Solid State Comput. Devices Circuits* 1, 58–66. doi: 10.1109/XCDC.2015.2448417
- Shukla, N., Datta, S., Parihar, A., and Raychowdhury, A. (2016). “Computing with coupled relaxation oscillators,” in *Future Trends in Microelectronics: Journey into the Unknown*, eds S. Luryi, J. Xu, and A. Zaslavsky (Hoboken, NJ: Wiley), 147–156. doi: 10.1002/9781119069225.ch2-3
- Shukla, N., Parihar, A., Cotter, M., Barth, M., Li, X., Chandramoorthy, N., et al. (2014). “Pairwise coupled hybrid vanadium dioxide-MOSFET (HVFT) oscillators for non-boolean associative computing,” in *Proceedings of the 2014 IEEE International Electron Devices Meeting*, San Francisco, CA, 28.7.1–28.7.4. doi: 10.1109/IEDM.2014.7047129
- von Neumann, J. (1957). *Non-linear Capacitance or Inductance Switching, Amplifying and Memory Devices*. U.S. Patent No. 2,815,488. Washington, DC: U.S. Patent and Trademark Office.
- Wu, J., Jiao, L., Li, R., and Chen, W. (2011). Clustering dynamics of nonlinear oscillator network: application to graph coloring problem. *Physica D* 240, 1972–1978. doi: 10.1016/j.physd.2011.09.010

Conflict of Interest: EC and SK were employed by the company IBM.

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

Copyright © 2021 Núñez, Avedillo, Jiménez, Quintana, Todri-Sañal, Corti, Karg and Linares-Barranco. 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.



Event-Based Trajectory Prediction Using Spiking Neural Networks

Guillaume Debat^{1*}, Tushar Chauhan¹, Benoit R. Cottureau¹, Timothée Masquelier¹, Michel Paindavoine² and Robin Baures¹

¹ CERC UMR 5549, CNRS—Université Toulouse 3, Toulouse, France, ² Laboratory for Research on Learning and Development (LEAD), University of Burgundy, CNRS UMR, Dijon, France

In recent years, event-based sensors have been combined with spiking neural networks (SNNs) to create a new generation of bio-inspired artificial vision systems. These systems can process spatio-temporal data in real time, and are highly energy efficient. In this study, we used a new hybrid event-based camera in conjunction with a multi-layer spiking neural network trained with a spike-timing-dependent plasticity learning rule. We showed that neurons learn from repeated and correlated spatio-temporal patterns in an unsupervised way and become selective to motion features, such as direction and speed. This motion selectivity can then be used to predict ball trajectory by adding a simple read-out layer composed of polynomial regressions, and trained in a supervised manner. Hence, we show that a SNN receiving inputs from an event-based sensor can extract relevant spatio-temporal patterns to process and predict ball trajectories.

Keywords: SNN, STDP, unsupervised learning, spiking camera, ball trajectory prediction, motion selectivity

OPEN ACCESS

Edited by:

Angelo Arleo,
Centre National de la Recherche
Scientifique (CNRS), France

Reviewed by:

Michael Schmuker,
University of Hertfordshire,
United Kingdom
Timoleon Moraitis,
Huawei Technologies, Switzerland
Hananel Hazan,
Tufts University, United States

*Correspondence:

Guillaume Debat
guillaume.debat@cnrs.fr

Received: 26 January 2021

Accepted: 27 April 2021

Published: 24 May 2021

Citation:

Debat G, Chauhan T, Cottureau BR, Masquelier T, Paindavoine M and Baures R (2021) Event-Based Trajectory Prediction Using Spiking Neural Networks. *Front. Comput. Neurosci.* 15:658764. doi: 10.3389/fncom.2021.658764

INTRODUCTION

The original aim of Artificial Neural Networks (ANNs) was to mimic human or even non-human brain processing. The learning and generalization abilities of ANNs have led to great advances, particularly in solving visual tasks (Rawat and Wang, 2017). However, the quest for performance has taken ANNs away from their original bio-inspired function, even if ANNs show good performances with neural activity correlated with human cortical activity (Schrimpf et al., 2018).

There is, however, another category of neural networks, called Spiking Neural Networks (SNNs). SNNs use spikes as signals between neurons, and in this respect, are closer to the brain than ANNs. The temporality of these spikes provides additional information (Van Rullen et al., 2005), making SNNs good candidates to deal with spatio-temporal stimuli. Moreover, since spiking activity is usually binary-coded and sparse (Van Rullen and Thorpe, 2001; Perrinet et al., 2004), processing in SNNs is highly power efficient (Rueckauer et al., 2017; Barrios-Avilés et al., 2018; Pfeiffer and Pfeil, 2018).

SNNs can be coupled with synaptic plasticity rules such as STDP (Spike Timing Dependent Plasticity), which are bio-inspired and unsupervised.

SNNs with STDP rule have been applied many times on image categorization tasks, in order to benchmark them against more common ANNs or other SNNs (Diehl and Cook, 2015; Kheradpisheh et al., 2018; Lee et al., 2018; Thiele et al., 2018). However, most of these studies used static images as stimuli, and thus, did not take full advantage of the above-mentioned benefits. In contrast, videos (or spikes from spiking cameras also called event-based cameras) are more suited for SNNs due to their spatio-temporal nature (Pfeiffer and Pfeil, 2018; Iyer et al., 2021). Recently, (Orchard et al., 2017) used event-based cameras and mimicked retinal saccades

to perform categorization tasks on standard datasets like MNIST. This allowed spike processing on more biologically plausible and more realistic data where a temporal dimension was induced directly by the motion saccade. The performance of the model was not only excellent (Lee et al., 2016), but even surpassed state-of-the-art ANNs on temporally occluded images (Moraitis et al., 2020).

In order to process motion, frame-based cameras are the most common way to acquire data. Frame-based processing is different from that of the retina. Typically, the camera output is synchronous, and processed by the ANN frame-by-frame. This processing then induces response time delays depending on the number of frames per second (FPS), motion blur, and data redundancy, resulting here again in unnecessary resource consumption.

However, the development of visual (Lichtsteiner et al., 2008; Posch et al., 2011; Brandli et al., 2014; Son et al., 2017), audio (Liu et al., 2014), and tactile (Taunyanov et al., 2020) event-based sensors brings them closer to biomimetics. These sensors only encode variations (of brightness, frequency, etc.) and are fully asynchronous, much like the retina. This allows sensors to generate extremely sparse data and to considerably reduce response latency (Farabet et al., 2012). These sensors make it possible to take advantage of all the benefits of SNNs. Indeed, a combination of SNNs with event cameras has been used to solve several tasks, such as object detection (Bichler et al., 2012), optical flow estimation (Orchard et al., 2013; Adams and Harris, 2014; Paredes-Valles et al., 2019), motion detection, etc.

These studies show that a bio-inspired system composed of an SNN driven by inputs from an event-based camera can learn, in an unsupervised manner, to optimally process spatio-temporal data.

In this study, we used a specific type of event camera, the “Neurosoc,” introduced in section Choice of the Event Camera. We recorded ball trajectories with the NeuroSoc and used an SNN to learn specific features of the trajectory (direction, speed, shape). Our objective was to test the accuracy of this setup in predicting the arrival point of the ball under various presentation times. We wanted to test if our network is able to anticipate the arrival point of the ball based on a snapshot of the trajectory, like sport experts do on the field, for example (Farrow and Abernethy, 2003).

A ballistic trajectory is constrained by physical laws, and based on these regularities. Humans can anticipate the arrival point of a moving object from information about the object's position, velocity and direction (Aglioti et al., 2008). Likewise, in this study, we aimed to decode the output of the SNN with polynomial regression. If after learning, neurons code for precise directions and speeds, it should be possible to accurately predict where the ball will fall from the SNN responses.

MATERIALS AND METHODS

The aim of this study was to predict the ending point of a ball's trajectory from an artificial visual system. Our pipeline consisted of (1) a “Neurosoc” camera which generates spikes from ball

trajectories, (2) a 3-layer SNN equipped with an STDP learning rule which progressively becomes selective to motion patterns and (3) a read-out layer which uses polynomial regressions to recover the ending point of the ball's trajectory. We further detail these three parts in the next sections. **Figure 1** provides a schematic overview of the artificial system.

Neurosoc

Choice of the Event Camera

Several models of event-driven cameras have already been proposed in the industry (Prophesee, iniVation, Insightness, Samsung, CelePixel) and operate mainly according to a pixel-to-pixel temporal difference. Although the performances of these devices are remarkable in terms of temporal frequency and dynamic range, they suffer from some crucial limitations in terms of bio-inspired modeling, namely the inexistence of spatial filters upstream of the spike-generation. Using bio-inspired models which capture various aspects of the visual system (Masquelier and Thorpe, 2007), these filters make it possible to reinforce the performances of spike-based analysis by introducing a bio-inspired component upstream of the spike-generation.

In parallel to this observation, different devices have appeared during the last few years which allow the generation of spikes from standard CMOS image sensors (Abderrahmane and Miramond, 2019; Admin, 2020; Spike Event Sensor, 2021). The main objective behind these cameras is, on one hand, to be able to integrate spatial filters upstream of the spikes generation, and on the other hand, to have sensors of different formats going, for example, up to 2M pixels (Caiman Camera, 2021) [usually the pixel-count of event-based cameras is rather limited (Lichtsteiner et al., 2008; Posch et al., 2011; Brandli et al., 2014; Son et al., 2017)].

In order to guarantee reliable integration of spikes, it is imperative that these image sensors work in global-shutter mode (instantaneous image acquisition), and with a short exposure time (in the order of few milliseconds). Such cameras are an intermediary between event-based sensors and frame-based cameras, and allow for spatial and temporal filtering with high FPS.

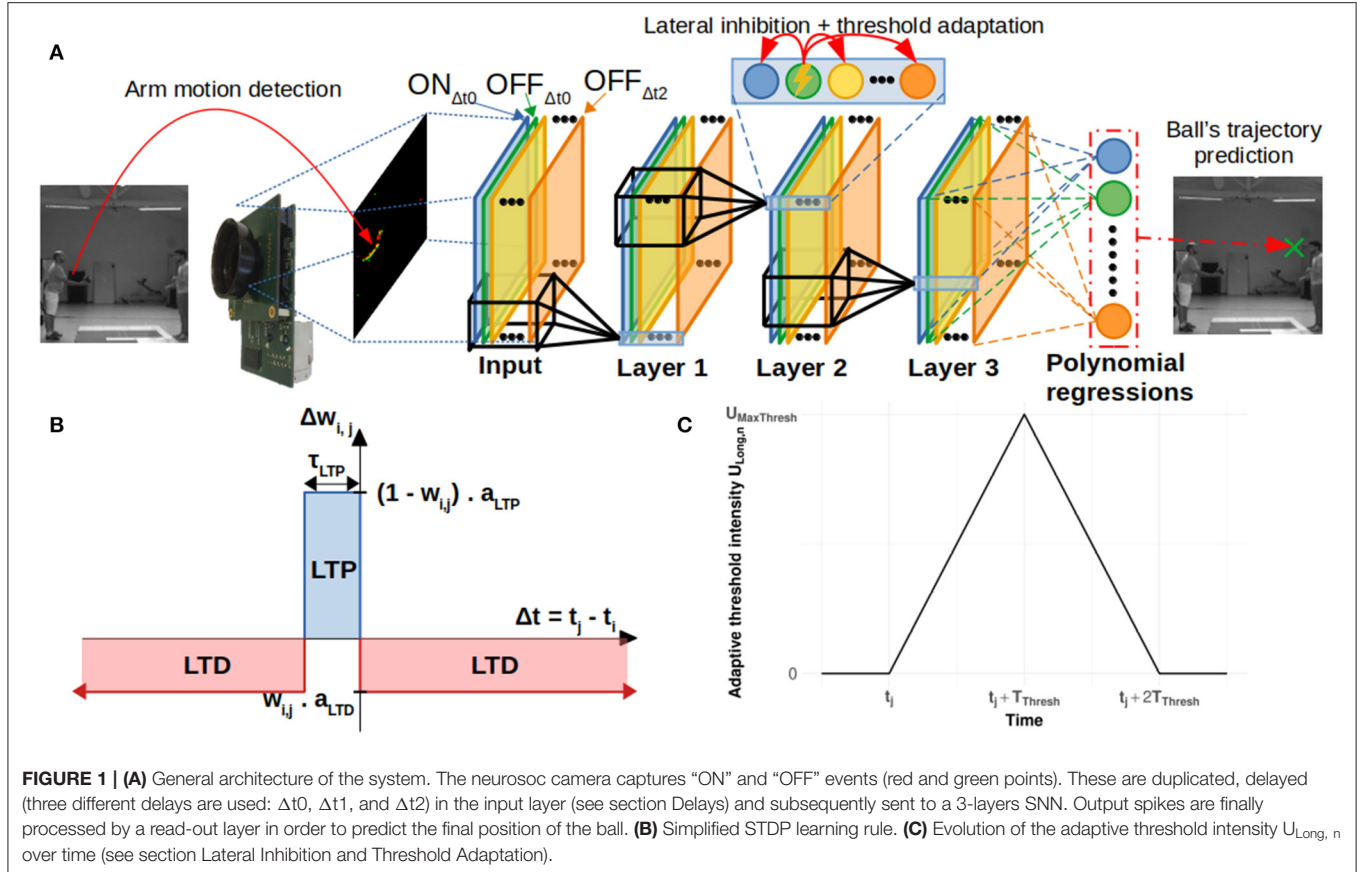
The NeuroSoc camera from Yumain (Spike Event Sensor, 2021) possesses these characteristics and was, therefore, chosen for this study. This camera operates at 240 frames per second at a resolution of 128×120 px. Spatial and temporal filters are embedded in a processing board closed to the image sensor (see section Architecture of the NeuroSoc Event Camera) to detect brightness variations and generate spikes. As explained above, these spatial filters (here DoG type) are closer to those found in the lateral geniculate nucleus in the human visual system. As a result, they reduce noise, detect edges, and increase output sparseness.

Architecture of the NeuroSoc Event Camera

We used the NeuroSoc event camera developed by Yumain which is based on a global-shutter CMOS MT9024 image sensor from On Semiconductor and a board called NeuroSoC. This board is composed of a MPSoC Zynq 7020 circuit from Xilinx and a 4 Gbits DDRAM memory (see **Figure 2**). The CMOS

TABLE 1 | Parameters for each SNN's layers.

	W _{Max}	τ_{memb}	N _f	τ_{LTP}	a _{LTP}	a _{LTD}	f _{Inst}	f _{Long}	T _{Thresh}
Layer 1	1.873	0.01	60	0.0754	0.00195	0.0005	3.0	2.89	0.031
Layer 2	0.813	0.052	80	0.0236	0.0131	0.00118	2.82	2.41	0.023
Layer 3	1.308	0.039	100	0.0368	0.00815	0.00198	3.46	1.9	0.051



sensor operates in global-shutter mode (instantaneous image acquisition) with an exposure time in the range of 31 ns to 4 ms. In the context of this study, images are generated in a 128×120 pixels format guaranteeing a throughput of 240 frames per second, with an exposure time of 3.7 ms due to low luminosity conditions (window shutters closed for proper operation of the Vicon). Images from the image sensor transmitted to the Zynq MPSoC circuit are filtered in real time in order to extract the salient parts of the objects contained in the images. The first step of the process consisted of calculating the difference between the images at time t_n and t_{n-1} (the sampling period $t_n - t_{n-1}$ was 4.17 ms or 1/240 fps). In this study, a DoG (Difference of Gaussian) filter was applied to this difference. The output of the filter was classified (positive/negative values generate ON/OFF spikes) and sorted according to the most important to the least important absolute values above a threshold, thus constituting a train of temporal spikes. The threshold value was set manually during the acquisition phase, and was adjusted to extract as many spikes from movements as possible while keeping the noise level

low. As shown in **Figure 2**, all these treatments are implemented in the FPGA within the Zynq MPSoC.

The spike stream was transmitted outside the camera *via* an ethernet link. Input/output management (spike transmission, sensor exposure time control) of the event camera was performed through the ARM processor of the Zynq MPSoC.

Time Encoding

The outputs of the filters implemented in the cameras (see above) were first thresholded. Values above threshold were subsequently converted into spikes using an intensity to latency conversion (Thorpe et al., 2001; VanRullen et al., 2005; Masquelier and Thorpe, 2007; Chauhan et al., 2018). Spike-latencies were obtained by inverting the output values of the corresponding filters. Spikes were spread between the current and the next frame, see Equation (1):

$$\Delta t_{s,rel} = \frac{(I_s - I_{min})}{FPS \cdot (I_{max} - I_{min})} \text{ and } t_s = \frac{F}{FPS} + \Delta t_{s,rel} \quad (1)$$

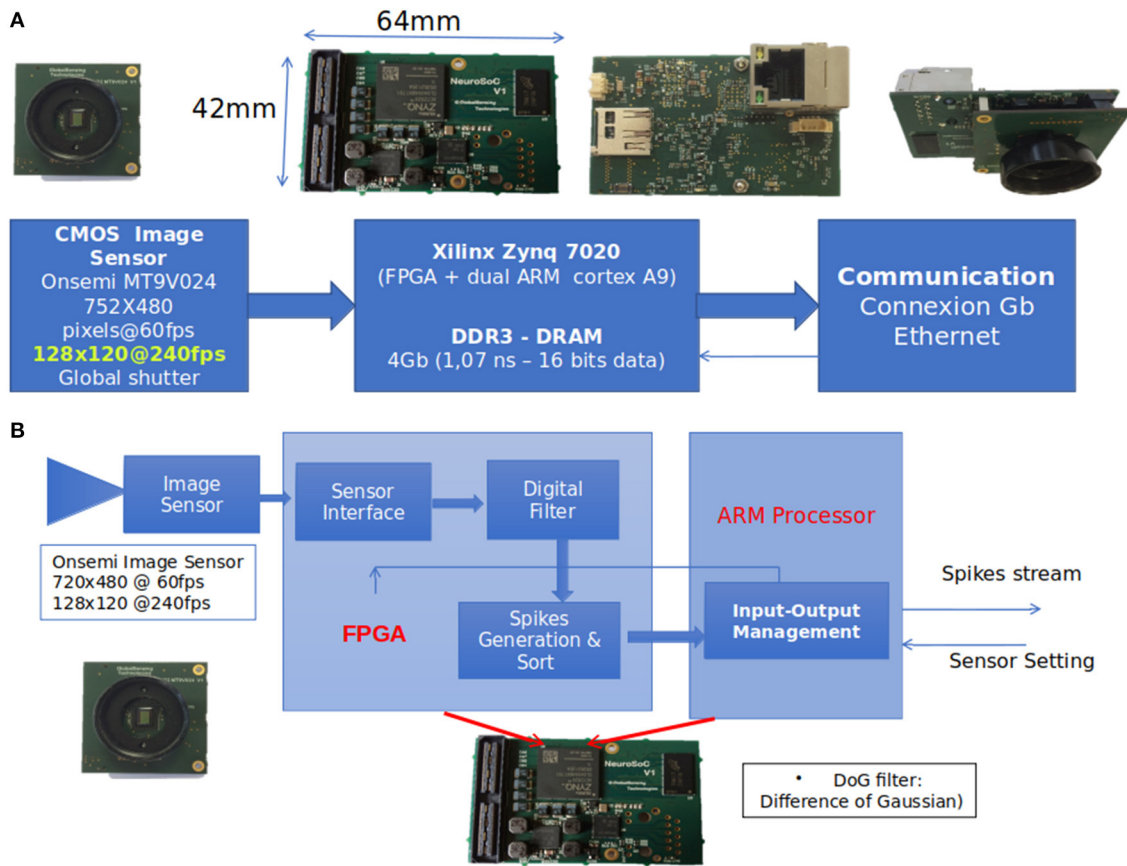


FIGURE 2 | (A) The event-based camera used in the study (NeuroSoC). It is based on a global shutter CMOS image sensor connected to a MPSoC Zynq circuit. **(B)** Implementation treatments on NeuroSoC board.

With:

F = Index of the current frame

$\Delta t_{s, rel}$ = Relative time of the spike s to the current frame F

I_s = Inverted intensity of the spike s

t_s = Time of the spike s .

SNN

In this study, the architecture we used is similar to other proposed multi-layer convolutional SNN (Masquelier and Thorpe, 2007; Tavanaei and Maida, 2017; Kheradpisheh et al., 2018; Mozafari et al., 2018; Thiele et al., 2018; Paredes-Valles et al., 2019). These studies highlighted the relevance of multi-layer SNN trained with an STDP learning rule to extract spatiotemporal features. Our architecture is composed of 3 layers trained with STDP. There is no pooling layer, and the network uses delays similar to (Paredes-Valles et al., 2019), but directly applied to the input layer generated by the NeuroSoC (see Figure 1A). We used a 3-layer SNN, composed of leaky integrate and fire (LIF) neurons with feedforward connections and lateral inhibition. The synaptic weights of the feedforward connections were learnt through a simplified STDP rule (Bichler et al., 2012). Similar as (Masquelier and Thorpe, 2007; Paredes-Valles et al., 2019),

we used a weight sharing process with retinotopically organized neurons connected to $5 \times 5 \times d$ patches and a stride of 1 (d corresponds to the number of convolutional filters in the previous layer). The network was also endowed with a lateral inhibition mechanism which reduced the membrane potential of all neurons sharing the same position, instead of resetting it (see details in section Lateral Inhibition and Threshold Adaptation). Simulations were performed using a C++ code developed by the team.

Neuron Model

Our SNN was based on leaky integrate and fire (LIF) neurons. When such a neuron receives an incoming spike, its membrane potential increases in proportion to the synaptic weight that connects it to the pre-synaptic neuron that emitted the spike. In the absence of incoming spikes, the neuron membrane potential leaks according to Equation (2):

$$U_i(t) = U_{rest} + (U(t_k) - U_{rest}) \cdot \exp\left(\frac{-(t - t_k)}{\tau_{memb}}\right) \quad (2)$$

t_k : last time update

U_{rest} : resting potential = 0

The SNN was event-based, and the membrane potential of a neuron was only updated when an incoming spike was received by the neuron. First, the leak was applied, and then, a value W_{ij} (weight connection between neuron i and j , constrained between 0 and 1) multiplied by W_{Max} was added to the membrane potential.

A given neuron emitted a spike when its membrane potential reached a threshold value U_{Thresh} . A spike was then generated and propagated to the next layer and the membrane potential was reset to its resting-state value U_{rest} .

STDP

To update synaptic weight connections and to give neurons the ability to learn specific features, we used a bio-inspired unsupervised learning rule called spike-timing dependent plasticity (STDP) (Bi and Poo, 2001; Caporale and Dan, 2008). It detects input correlations and enables the neurons to become selective to most frequently occurring patterns. Previous studies have already demonstrated that SNNs equipped with STDP can learn repetitive specific patterns (Masquelier et al., 2008), visual properties such as orientation (Delorme et al., 2001; Masquelier, 2012), binocular disparity (Chauhan et al., 2018) or shape (Masquelier and Thorpe, 2007; Diehl and Cook, 2015; Thiele et al., 2018). In this study, we used a simplified STDP learning rule inspired from (Bichler et al., 2012) (see **Figure 1B**). This simplified version does not include a time window for LTD. If the time of the last presynaptic spike is not included within the LTP window (τ_{LTP}), LTD is applied. Synaptic weight updates corresponding to spikes occurring during the LTP window are all equal and therefore independent of spike times, as shown in Equation (3). Contrary to the synaptic update rule used in (Bichler et al., 2012), we included a multiplicative term which depends on the current weight. This multiplicative rule also forces a soft-bound on the weights in the interval (0, 1).

$$\begin{aligned}\Delta W_{ij} &= (1 - W_{ij}) * a_{\text{LTP}} \text{ if } t_j - t_i < \tau_{\text{LTP}} \text{ and} \\ \Delta W_{ij} &= -W_{ij} * a_{\text{LTD}} \text{ otherwise}\end{aligned}\quad (3)$$

W_{ij} : weight synaptic connection between afferent neuron i and j
 t_i : last spike of afferent i .
 t_j : last spike of neuron j .
 $a_{\text{LTP}}/a_{\text{LTD}}$: amplitude of the potentiation/depression
 τ_{LTP} : LTP time window.

Delays

Although SNNs equipped with STDP can learn different spatial patterns (orientation, spatial frequency, binocular disparity, ...), learning motion direction is a harder task because it relies on spatio-temporal properties. Indeed, input neurons with similar spatial positions spike but not in the same order: for leftward (or rightward) motion, inputs placed on the left (right) spike first, followed by other inputs from the left (right) to the right (left). The common way to improve SNNs' temporal selectivity and thereby permit discrimination of motion direction is to use delays. These delays improve synchrony between input spikes for specific spatio-temporal patterns (rightward motion, for example) and desynchronize inputs for opposite patterns (leftward motion).

In order to work with delays, it is necessary to be careful about the strategy employed to learn them. Over the last few years, various approaches have been proposed to select these delays (Eurich et al., 2000). For instance, the *delay shift* (Eurich et al., 2000; Tversky and Miikkulainen, 2002; Gibson et al., 2014) approach consists of learning the delay from the input spikes. In addition to the weight-learning rule, this approach also uses a specific learning rule for delays to increase the simultaneity of the input spikes. Another approach, called *delay selection* (Paredes-Valles et al., 2019), consists of duplicating synapses and adding delays to them. In delay selection, a single weight-learning rule can be used to select synapses with appropriate delays. In this study, we used this second method (Eurich et al., 2000; Paredes-Valles et al., 2019). Three different delays (0, d_1 , and d_2) were applied on each input and each synapse was therefore duplicated three times. Because we consider both "on" and "off" events, there were 6 different synapses for each pixel-positions ($\text{ON}_{\Delta t=0}$, $\text{OFF}_{\Delta t=0}$, $\text{ON}_{\Delta t=d_1}$, ..., $\text{OFF}_{\Delta t=d_2}$) as shown in **Figure 1A**.

Lateral Inhibition and Threshold Adaptation

When a neuron spiked, the neurons of the same layer sharing the same retinotopic position were prevented from spiking. We used an instantaneous inhibition, the membrane potential was reduced by a specific value U_{Inst} [see, e.g., Diehl and Cook (2015) and Delorme et al. (2001)]. The purpose of this inhibition was to prevent the filters from learning similar patterns. It therefore increases selectivity and sparsity within the neural population.

We also used a homeostatic process which added a penalty $U_{\text{long,n}}$ to the neurons by increasing their threshold. The magnitude of change in the threshold had an inverted "V" shape in time—it increased and subsequently decayed back to zero (the increase and decrease both lasted for a time-interval T_{Thresh}), allowing a time dependent threshold adaptation (see **Figure 1C** and **Annex 1** in Supplementary Material).

The threshold variation and inhibition intensity was made proportional to U_{PropInh} : the current average of squared membrane potentials of neurons connected to the same patch (see **Annex 1** in Supplementary Material).

Training Procedure

The 3-layer SNN was trained layer-by-layer in an unsupervised manner using the STDP learning rule described in section Delays. Seventy percentage of the recorded trajectories were used for training. During the training and testing phases, trajectories were presented individually and were separated by periods of 2 s without spikes, allowing the SNN to reset to its initial state. The third layer output was then used to make predictions, as detailed in section Trajectory Prediction. For each convolutional filter ($N_f = 100$ for the third layer), a polynomial regression was performed with the spatial position (x and y) of the spiking neuron as input and the ball's vertical position as the predicted variable.

An SNN with STDP learning rule has some parameters which need to be tuned (membrane potential time constant, ratio LTP/LTD, learning rate, ...). We estimated these parameters with a genetic algorithm (Mohammadi et al., 2017) (see **Table 1**). A simplified version of the prediction process presented in section Trajectory Prediction was used. The cost function for the

minimization was the average value of the mean prediction error at 15, 45, ..., 90% of trajectory presentation.

Data Acquisition and Comparison

We recorded 297 passes of the ball between two participants using the Neurosoc camera. The two participants were separated by a distance of ~ 2.30 m, and the trajectories included multiple velocities. The camera's direction was perpendicular to the trajectories.

Each of the 297 trajectories was manually labeled so as to determine the start and end points of the throw, the direction and the height of the reception (X and Y-axis, respectively, see section Trajectory Prediction). The trajectories were partitioned using a 70–30 train-test split, and the final training and testing sets contained 208 and 89 trajectories, respectively.

To quantify the neuron's selectivity, we recorded the ball's 3D spatial position with Vicon cameras (Merriault et al., 2017). These cameras use infrared light to capture the position of markers placed on the ball, at a frequency of 200 Hz. Four targets were placed at four opposite positions on the ball. The ball's center of gravity was obtained by averaging these positions. A low-pass filter was applied to these values with a cut-off frequency of 6 Hz. Using the exact value of the ball over time, we could easily determine the ball's direction and speed.

Vicon technology uses infrared flashing lights to detect targets. These lights were perceived by the camera and generated big variations of brightness which generated spikes. To overcome this issue, we used an anti-IR lens with a cut-off wavelength of 730 nm (SP730 Near-IR/Colorless Dichroic Block Shortpass Filter, 2015).

Trajectory Prediction

Our main objective was to predict the ball's reception point from the output of the SNN. Since this point would be highly impacted by the receiver (depending on whether he moved his arms forward to intercept the ball or not), we restricted these predictions to the y-axis. The prediction along the x-axis was simplified, with only two choices reflecting the ball's direction (right or left).

The y-value was the ball's position when it crosses the blue/brown line for leftward/rightward directions (see Figure 3B). Polynomial regressions (PR) for each filter of the last layer were used to decode the SNN output and predict the y-value. We used a simple decoder to ensure that performances were mostly driven by processing within the SNN. More complicated decoders could provide better results, but this study focuses on the performance of the SNN. Second degree polynomial regressions were chosen because filters of the SNN spiked for specific patterns. If the SNN did not develop any motion direction selectivity, predictions from PRs would be inaccurate. To demonstrate this point, we also applied PRs to the outputs of the neurosoc camera (i.e., PRs were performed on the SNN inputs) (see Figure 6).

In this study, to determine the prediction on the Y-axis, we used a PR of second degree for each filter n , Equation (4):

$$Y_{Pred,n} = a_{00} + a_{10}x + a_{01}y + a_{20}x^2 + a_{02}y^2 + a_{11}xy \quad (4)$$

With x and y , the spatial position of the spiking neuron of filter n , as schematized in Figure 3A. PR's parameters were learned by presenting all spatial positions of spiking neurons of corresponding filter n as PR's input and the corresponding y-value at the spike time as value to predict. This process was applied for each filter of the output layer.

The ball's predicted direction (X_{pred}) was not taken into account during the PR. Filters mostly encode for specific directions. The direction was thus independent of the PR. The root mean square error (RMSE) of each PR was then computed in order to evaluate the reliability of each PR. Finally, a scoring mechanism (see Figure 3C and Annex 2 in Supplementary Material) was used to spatially integrate the predicted value $Y_{Pred,n}$ based on the PR's reliability, and perform an average prediction over time.

Unsupervised Motion Tracking

The motion selectivity of filters should allow us to track specific motions of stimuli based on their speed, directions, shape, etc. There are few different motions-patterns in the stimuli, mainly the ball and arms which can be divided into multiple parts (hands, forearm, ...). An ideal way to evaluate our network's ability to track specific motions should be to label all of them, but this would make the task highly time-consuming and unfeasible. This study mainly evaluates if filters spike for the ball or for shapes similar to the ball.

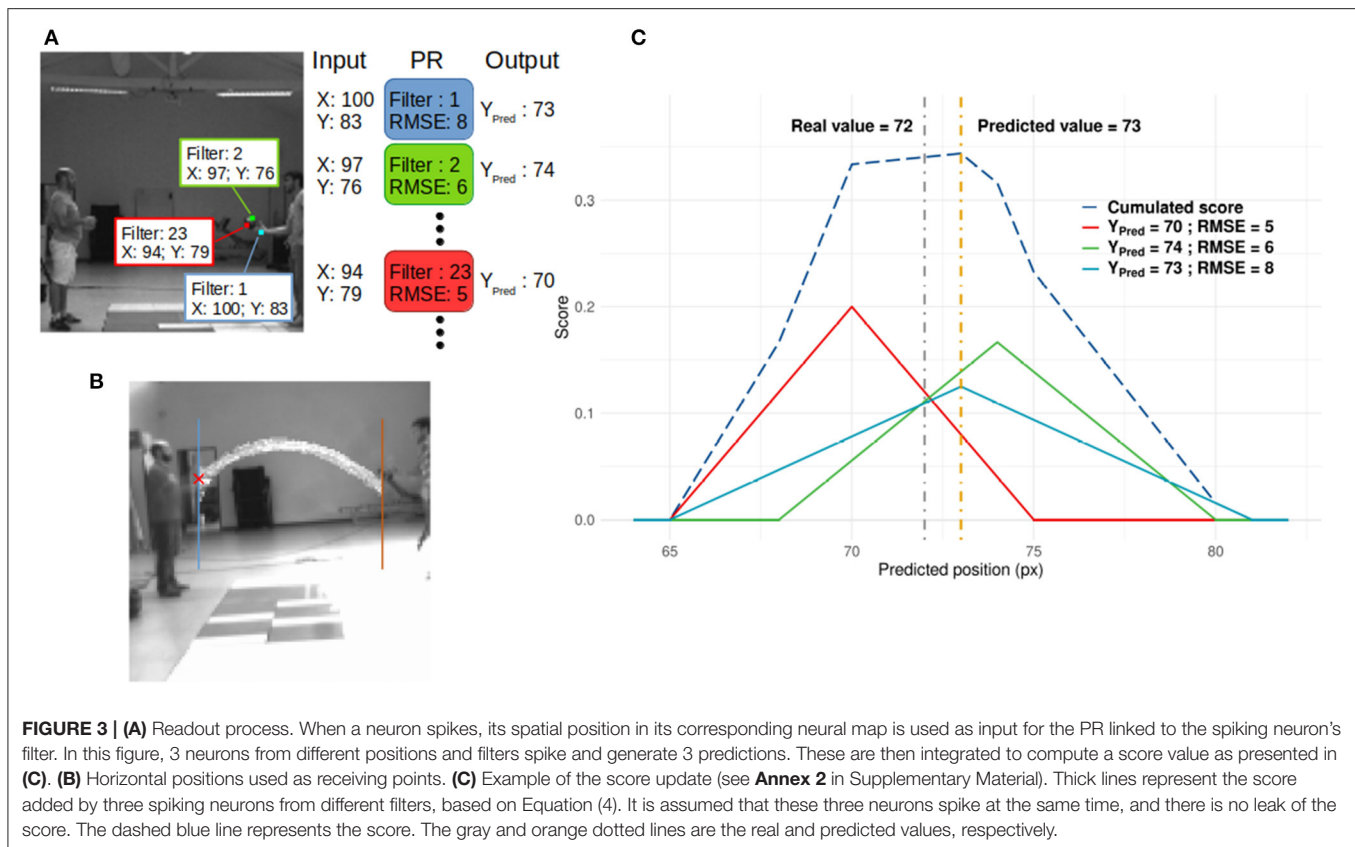
A first way to determine whether a filter spikes for the ball is to compute the distance between the center of the ball and the spike's position. Because the camera was facing the plan of the ball trajectory, there is a linear relation between the ball's position in the reference frame of the camera and the one of the Vicon. So, given the ball's position in the reference frame of the Vicon, we can easily compute its position on the frame and compute the distance between a given spike and the center of the ball.

Then, by looking at the mean distance D_n from the ball of all spikes for each filter n , we can determine if the given spike encodes the ball's motion (if D_n is very low) or some other feature like a part of an arm.

Because a neuron is determined by its position and its filter, which encodes or not for ball motion (based on D_n), we should be able to track the ball's motion based on the output of this network. As an example, in the context of Figure 3A, we can expect that filter 2 and 23 could be used to track ball motion and filter 1 for hand motion.

Comparison With Human Performance

In order to compare the performances of our system to human capabilities, we conducted an experiment during which 12 participants (mean age = 27 ± 12.9) were instructed to predict the end-point of the ball at different time steps. The NeuroSoc camera also recorded the classical frame-based video at $f = 240$ fps in parallel with spike estimation and transmission. The trajectories presented to these participants were therefore exactly the same as the ones used to test our SNN. All participants had normal or corrected-to-normal vision, and were healthy and without any known oculomotor abnormalities. Participants were naïve with respect to the purpose of the experiment, which



received the appropriate ethical authorization from the “Comité d’éthique de la Recherche” of the Federal University of Toulouse (agreement 2020-279). The sample size was determined using G*Power (Faul et al., 2007) after having analyzed the results of a previous experiment investigating the influence of presentation duration on anticipation’s performances. The results showed that for a desired power of 0.90, a total sample size of 12 participants was required.

Only a part of the trajectories (i.e., the first 15, 30, 45, 60, 75, and 90%) were presented to the participants who were instructed to predict the end-point by clicking with the mouse on the anticipated ending point, without any temporal constraint. The experiment was divided into three parts:

- Pre-learning: first time videos are presented to participants
- Active-learning: after each prediction made by the participant, the exact arrival point was -shown on the screen to provide the prediction error to the user, as a feedback
- Post-learning: same procedure as Pre-learning (no error was shown) but subjects had the experience of the “Active-learning” phase.

For each part, 8 trajectories were shown for each percentage of presentation time, for a total of 48 trials in each condition. Videos with a dimension of 384×360 pixels were presented on a 13.3-in. screen (60 Hz, full resolution $1,366 \times 768$, dimension 29.5 \times 17 cm in horizontal by vertical).

In contrast to our SNN, the human participants already had experience with ball motion or motion in general. We

nonetheless included an active-learning phase in the experiment so that participants could adapt to its specificities.

RESULTS

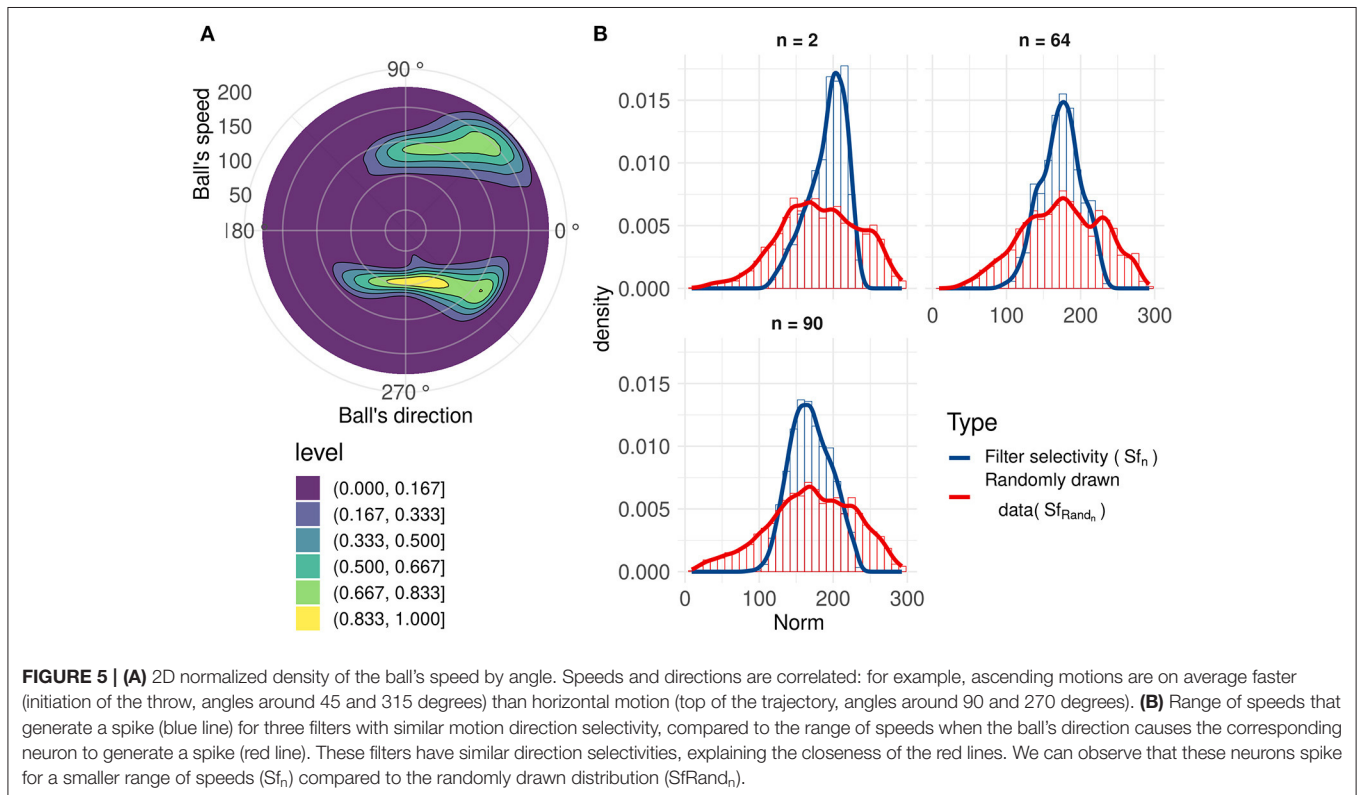
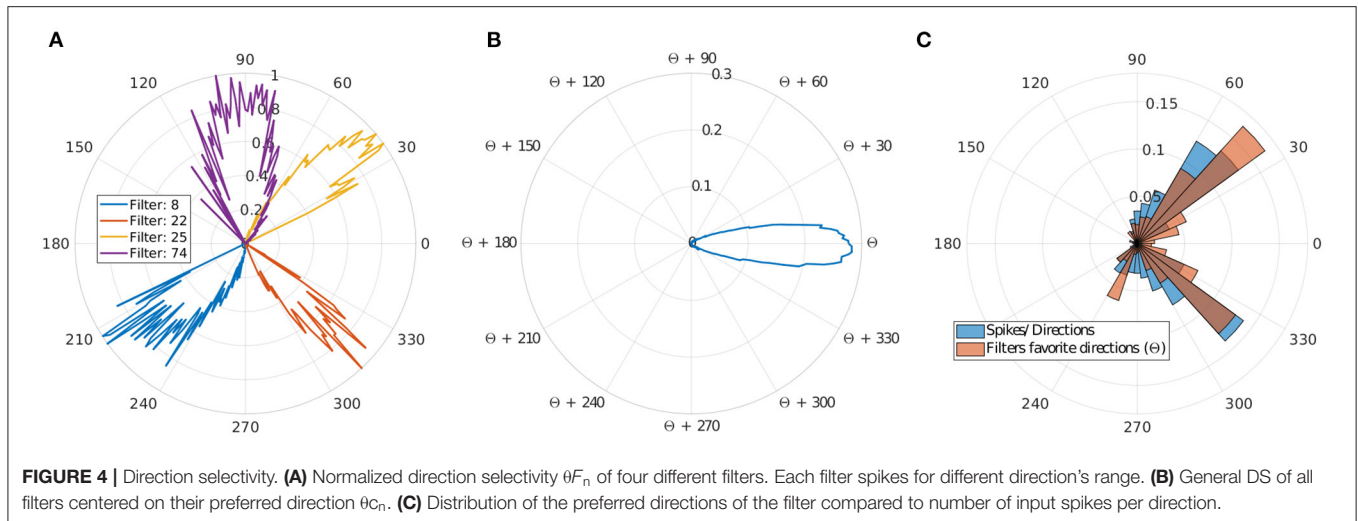
Selectivity

During the acquisition of the videos, we used the Vicon technology to precisely measure the trajectory and position of the ball (see section Data Acquisition and Comparison). After the learning phase, we characterized the selectivity of the neurons from these ground truth data. We averaged all results over 6 simulations of the SNN with different random weight-initializations. The network learned using the training set, and the results below are the analysis from the output of the third layer with the test set as input of the SNN.

Direction Selectivity

Motion direction selectivity was obtained by counting the number of spikes from the output layer triggered by all the ball’s trajectories. For each filter n , we divided the number of spikes for each direction of the ball (rounded to the unit) by the number of occurrences of each of these directions during all presented throws, giving us θ_{fn} . These results are presented in **Figures 4A,B**.

We can observe that filters mostly spike for directions similar to their preferred direction θ_{cn} , which demonstrates a strong motion direction selectivity. **Figure 4C**, provides a



comparison between a histogram of the preferred directions θ_{c_n} of our filters and the occurrence of input spikes for each direction.

Different filters are selective to different trajectories as shown in **Figures 4A,C**, and there are more filters selective to ascending motion. Indeed, leftward/rightward rising directions (which include the throw phase, when the ball is still in the thrower's hand) represent the majority of our trajectories, and this result confirms the ability of our model to learn the direction selectivity patterns in the inputs.

Speed Selectivity

Using an analysis similar to direction selectivity, we evaluated Sf_n : the speed distribution for which each filter n spikes. However, this is not sufficient to evaluate the speed selectivity as there is a broad range of different speeds of the ball for all trajectories, but given the limited set of recorded trajectories, directions and speeds are correlated (kinematics in a uniform gravitational field), as shown in **Figure 5A**.

To better characterize speed selectivity, we evaluated it independently of its direction selectivity. We compared Sf_n

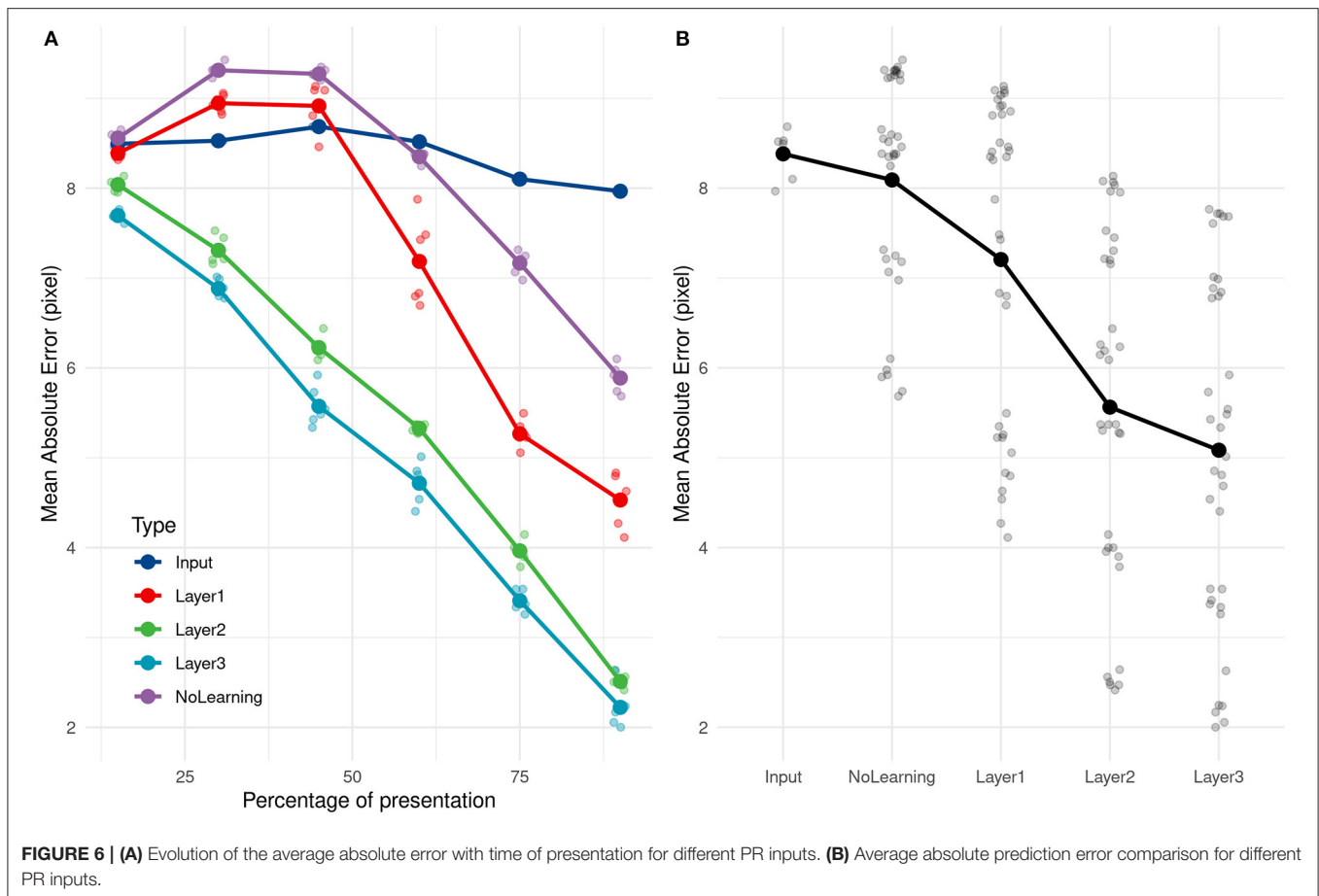


FIGURE 6 | (A) Evolution of the average absolute error with time of presentation for different PR inputs. **(B)** Average absolute prediction error comparison for different PR inputs.

with a randomly drawn distribution $SfRand_n$ of ball's speeds with directions similar to the filter's direction selectivity θf_n . Details about this randomly drawn distribution and filters' speed selectivity can be found in **Annex 3** in Supplementary Material. A non-selective filter should have an Sf_n very close to $SfRand_n$. **Figure 5B**, shows that filters become selective to the ball's speed. These three filters with similar direction selectivity spike for a smaller range of speeds (Sf_n) than the randomly drawn distribution ($SfRand_n$), and the distributions also have different peak values. These results highlight filters' selectivity for a range of speeds and confirms previous results (Paredes-Valles et al., 2019). This selectivity is broader than direction selectivity, and although we cannot determine speed with high precision, it still provides information about velocity, which is useful to make predictions about the trajectory of the ball.

Trajectory Prediction

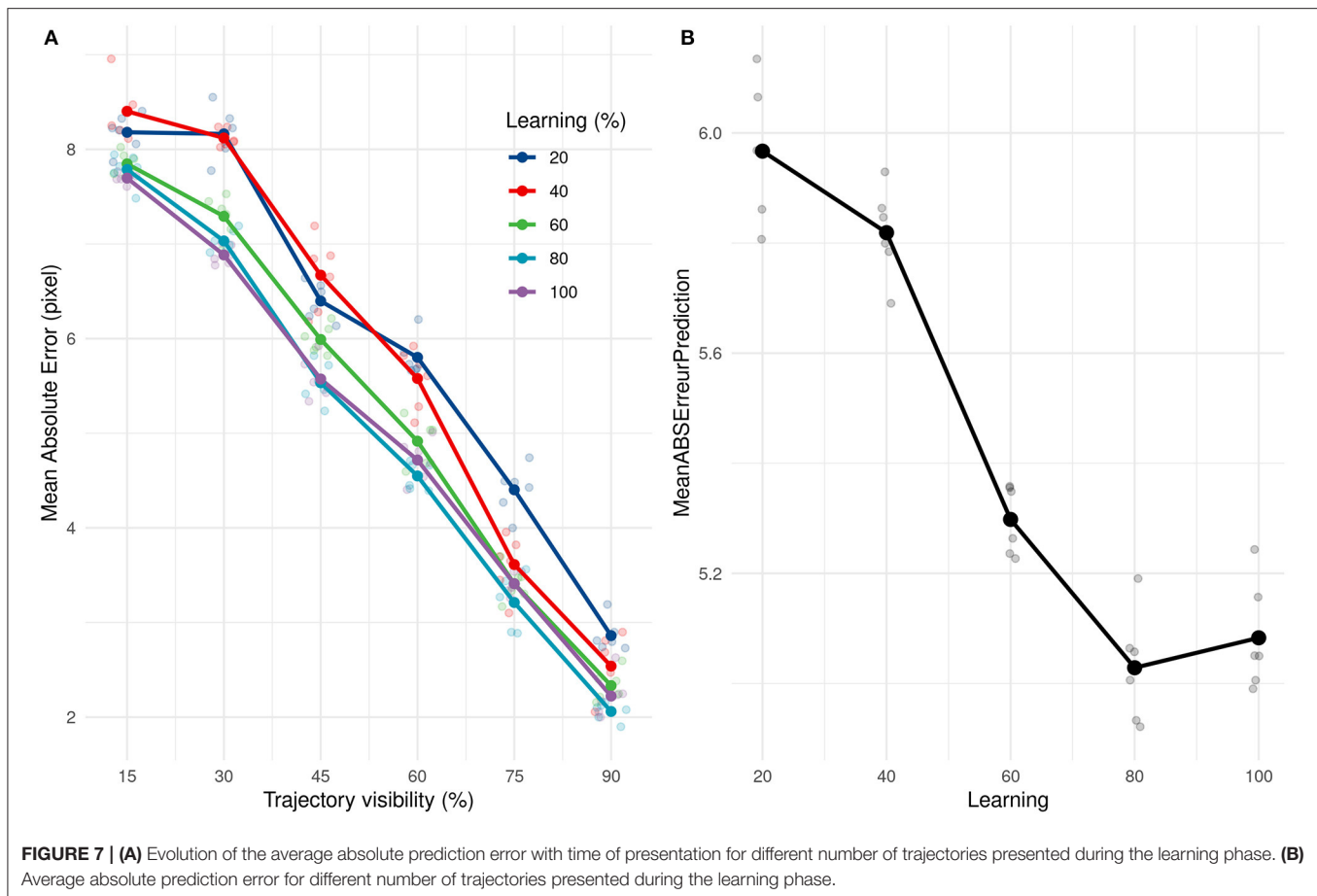
After training of the SNN and PRs, the test set of trajectories was presented. For each spike generated by the last layer, the prediction was updated. We analyzed the mean Absolute Error (AE) and variability of AE (SD AE) obtained during the test phase, through the mean of separate ANOVAs, with Visibility (15–90%) as a within factor.

The ANOVA on AE showed that Visibility influenced the mean AE, $F_{(5, 25)} = 1156.38$, $p < 0.001$. *Post-hoc* tests using

Bonferroni corrections demonstrated that the mean AE was always reduced as the visibility increased (see the layer 3 in **Figure 6**). We obtained good performances even with 15% of the trajectory with an error of 7.7 pixels. This error decreased with presentation time and went down to 2.2 pixels for 90% of trajectory presentation. No direction (rightward/leftward throw) error was made, whatever the percentage of presentation. It is important to note that for the lower Visibility condition (15%), the video is stopped on average 0.114s after the throw's initiation.

As a comparison, if we compute the mean of all training values, equals to 68.2 pixel, and use it as a naive predictor, we get an error of 9.4 pixels. Even with 15% of the trajectory presentation, the SNN can predict the throw's direction and is 16.7% better than this « naive » predictor. The ball was still in the thrower's hand for all trajectories at 15% of trajectory presentation and also at 30% for 35 over 89 trajectories (the ball had just left the hand for others).

The analysis of SD AE also showed a significant influence of the Visibility, $[F_{(5, 25)} = 143.60, p < 0.001]$, with the SD AE decreasing as the visibility increases. The SD goes from 5.605 pixels at 15% to 1.996 at 90%, with no significant difference between the 15 and 30% conditions, and 75 and 90% conditions. The evolution of the SD AE depending on visibility can be seen in **Figure 10B**.



We compared the prediction error over different layers, with an untrained network and with direct inputs to evaluate the performance of the learning and the impact of adding layers to the network. As shown in **Figure 6**, using the direct input (i.e., output of the camera), we cannot predict the reception point. As expected, PRs are not accurate with just the position information, and need more complex features, such as speed and direction encoded in filters.

Error prediction decreased over the successive layers. The first layer still made some errors but performed far better on the untrained network, more specifically for the end of the trajectory.

We subsequently investigated how quickly our solution (SNN + PR) could learn and how many presentations it needed to provide correct estimations. We applied the same learning and prediction process as before, but the learning was done with only a subset (20, 40, 60, 80, and 100%) of the 211 training trajectories. Our approach led to very good performance even when only learning from 20% of trajectories. Performances increased and then reached a ceiling at 80%, ~168 trajectories, as shown in **Figure 7**.

Unsupervised Motion Tracking

Numerous studies showed that SNNs equipped with STDP develop a progressive selectivity to shape along their hierarchy

(as more conventional neural networks) (Masquelier and Thorpe, 2007; Kheradpisheh et al., 2018; Thiele et al., 2018). Neurons in the first layers are selective to edges whereas neurons in deeper layers are selective to more complex features. In the context of our SNNs, filters with a mean distance from the ball under 6 pixels mostly encode for features related to the ball (including the forearm of the thrower when the ball is still in their hands). Filters with an higher value encode for different features, like other parts of the arm, the receiver, etc. The distribution of D_n highlights the ability of filters to encode for specific motions' patterns, such as the motion of the ball (see **Figure 8A**). As shown in **Figure 8B**, we can see the position of spikes for 4 different filters. Each filter spikes for specific positions and different motion patterns. This unsupervised selectivity could then be used to track motion of specific objects such as the ball, the thrower's hand, etc.

Human's Performances

We analyzed the performance of human participants (AE and SD AE) with two separate ANOVAs, with Learning (Pre-test, Active and Post-test) and Visibility (from 15 to 90%) as within-subject variables. The results show that visibility [$F_{(5, 55)} = 31.85$, $p < 0.001$] and learning [$F_{(2, 22)} = 10.78$, $p < 0.001$] have an influence on prediction error and there are no interactions between these two factors [$F_{(10, 110)} = 2.32$, $p = 0.016$]. Prediction

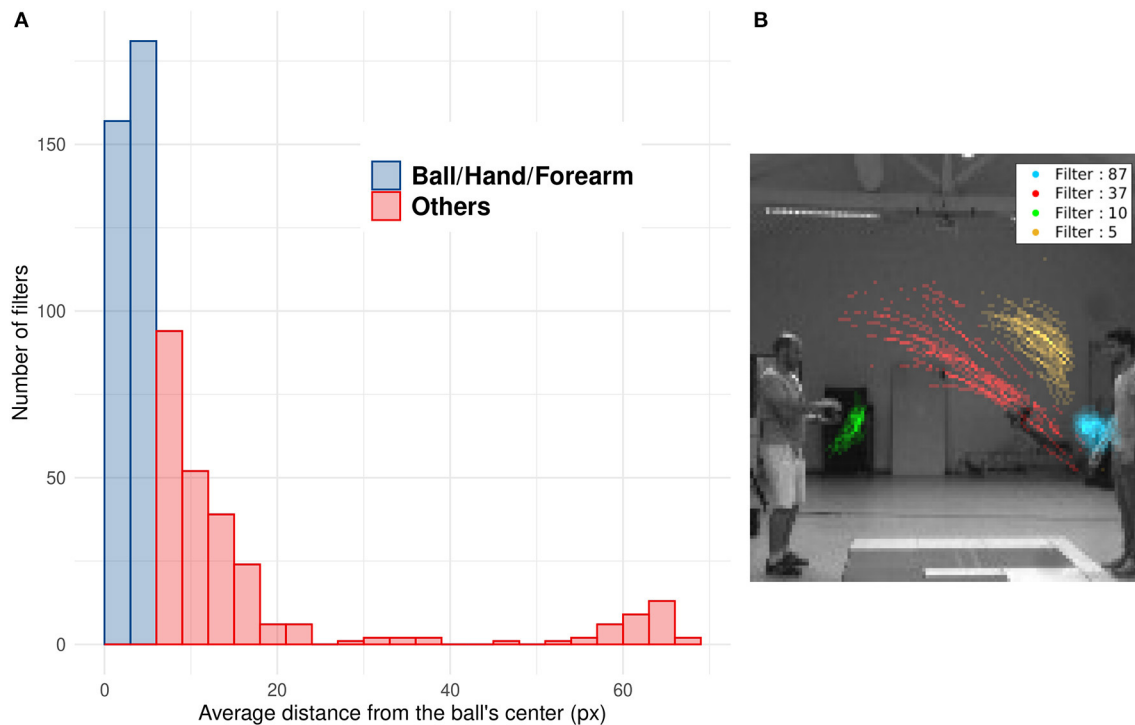


FIGURE 8 | (A) Histogram of the mean distance D_n of all filters. Blue bars represent filters with D_n under 6 pixel which mostly spike for the ball, the hand or the forearm. Red bars represent filters which spike for other features less correlated with the ball motion (arm, after throw motion, etc.). **(B)** Spike's position for four different filters: The filter number 37 (in red) spikes for ascending leftward ball's motion, the opposite of filter 5 (in yellow) which spikes for descending rightward ball's motion. Filter 87 (in blue) is selective to the right thrower's arm during the throwing motion (i.e., when the arm rises) and filter 10 (in green) to the left thrower's arm after throwing motion (i.e., when the arm goes back to initial position).

error decreases with the percentage of trajectory that is shown to the participants, as shown in **Figure 9B**. From *post-hoc* tests, we did not observe significant differences between 15 and 45 and between 45 and 60 percent of trajectory's presentation. There are also significant differences between pre-learning and other learning phases (learning and post-learning) and not between learning and post-learning, as shown in **Figure 9A**. These results highlight the effect of the learning phase which improves the prediction results which remain stable during the post-learning phase.

Performance Comparison

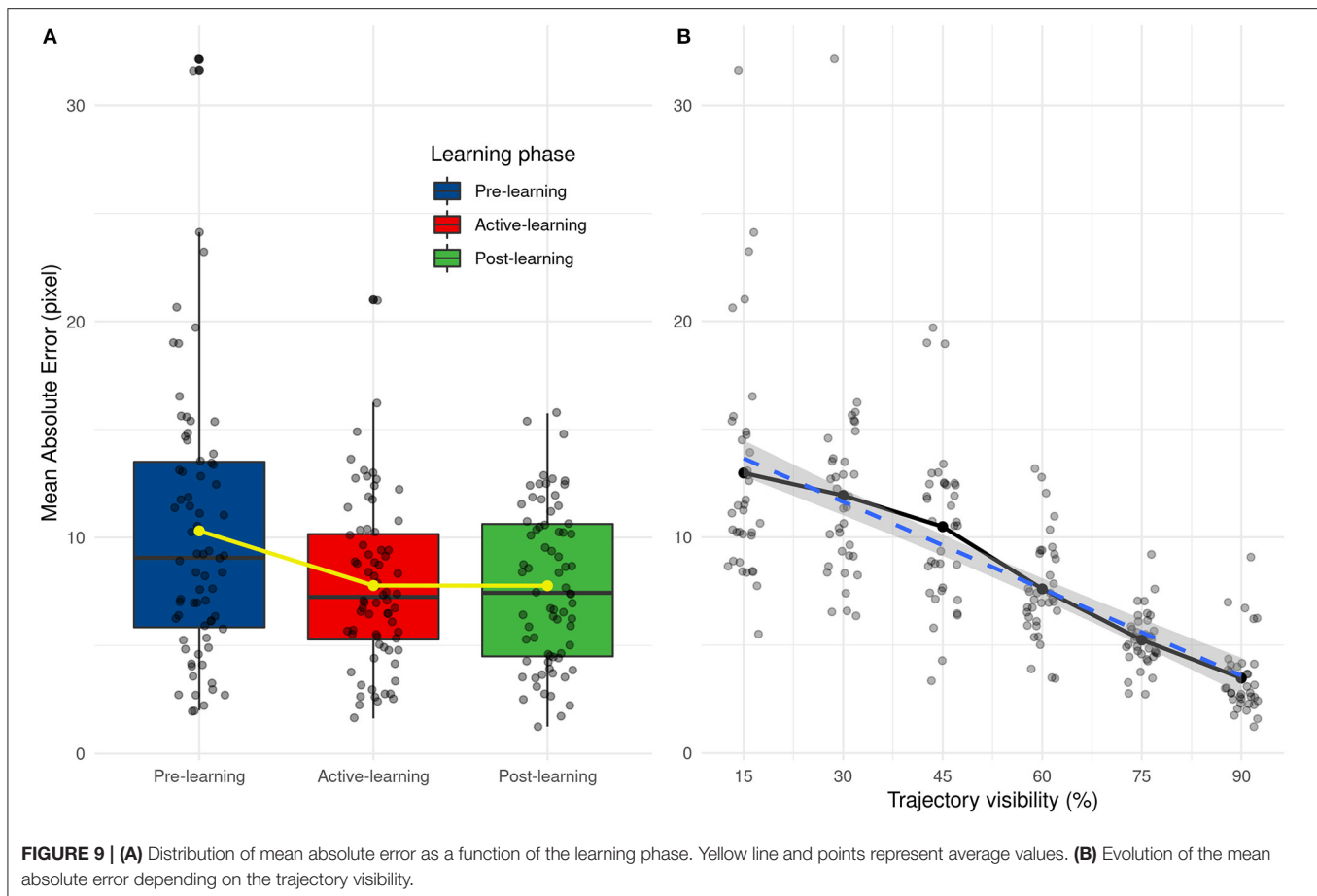
To compare the performances between the human participants and our SNN, we performed an ANOVA on the mean AE and SD AE for the trained condition only (i.e., Post-Test condition for the humans and test set for the 6 simulations of the SNN). We used the participant-type (humans vs. SNN) as a between variable, and Visibility as a within variable.

By comparing the results between humans and SNN, a large difference between them can easily be seen at first when comparing their mean absolute error [$F_{(1, 16)} = 47.09$, $p < 0.001$] and their SD AE [$F_{(1, 16)} = 20.16$, $p < 0.001$] for each participant/simulation. This suggests that predictions made by the SNN are more accurate and more precise than human predictions.

The mean and SD AE decreases in a linear way as Visibility increases [for the mean: $F_{(5, 80)} = 38.21$, $p < 0.001$, for the SD: $F_{(5, 80)} = 22.88$, $p < 0.001$], as shown in **Figure 10**. Finally, there was a significant interaction, [$F_{(5, 80)} = 2.53$, $p = 0.036$] for AE only, indicating that the SNN always outperformed human participants, except when the trajectory was presented for 60 or 90% of the trajectory. The absence of the interaction regarding SD AE [$F_{(5, 80)} = 1.19$, $p = 0.32$] indicates that SD AE is generally lower for the SNN, independent of the visibility.

Next, we investigated how the SNN or human participants predict the ending point of the ball trajectory. The prediction strategy used by humans and the SNN is different as shown in **Figure 11**. Indeed, the SNN has mean predictions close to 68 pixels which is close to the average prediction value of the training set (equals to 68.2 pixels) and remains stable over time. On the contrary, average prediction for human participants varies over presentation time. On average, human participants under-estimate the final position of the ball (prediction under 68.2 pixels) for the beginning of the trajectory, which changes to a light over-estimation with time.

The variability of predicted values is also different between humans and our system. Indeed, the standard deviation of predicted values increases over presentation's times for our solution. This one is low for the beginning of the trajectory



and so our solution predicts values close to 68 px. Then, it increases the prediction's variability with more information and so presentation's time. This tendency is also valid for humans but less « significant » than our system.

These results show a different way to predict the reception point between our solution and humans. On the one hand, our solution makes cautious and « statistical » predictions by targeting close on average to 68.2 px (the average value of the training set). As visibility increases, the SNN is able to step aside from this mean value and predict a different position, close to the correct value. In other words, the SNN makes cautious predictions, based on the average value of the dataset when the visibility is low, but is able to make more liberal—but accurate—predictions as visibility increases. Humans however seem to act differently, without using a statistical mean as a target. Their perception is variable even under low visibility conditions, indicating a trial-by-trial decision, and seems to switch from an underestimation to a small overestimation as visibility increases.

DISCUSSION

Studies have already shown the ability of SNNs to process motion from a spike-based visual flow. This study extends the evidence

of the reliability of using SNNs for motion processing and the efficiency of such networks.

One of our study's main contributions is the new sensor used to generate spikes and to analyze motion. Usually, SNNs are fed by asynchronous spiking cameras (Bichler et al., 2012; Orchard et al., 2013; Paredes-Valles et al., 2019) which spike for each step of brightness, thus generating a large number of spikes.

In this study, the brightness variation is encoded in the spike's temporality using the neuroSoc and an intensity to latency conversion rule. The filtering process used by the Neurosoc sensor thus provides less noisy and sparser information.

Firstly, we show our system is able to process motion information from a spiking camera, which is reliable for making ball trajectory predictions.

Secondly, we show that the selectivity of the filters can then be used to track specific motion patterns (arm, ball, etc.). Like the motion selectivity of our network, the tracking ability is fully unsupervised. While it was not the main objective of this study, it highlights the ability of unsupervised neural networks to solve multiple tasks. Thus, we can expect it to be relevant to other related visual tasks such as gesture recognition, counting tasks, object recognition, etc.

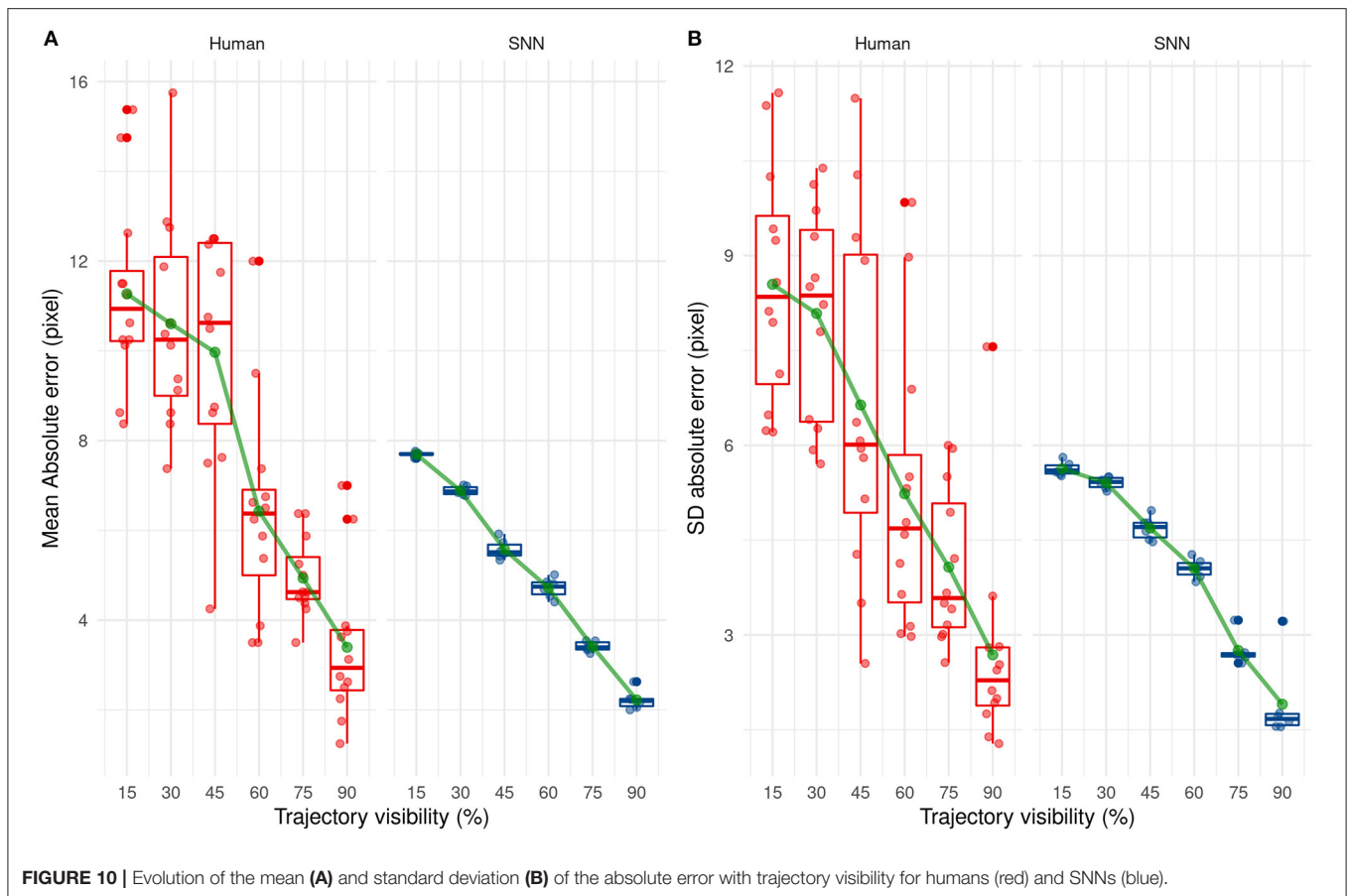


FIGURE 10 | Evolution of the mean (A) and standard deviation (B) of the absolute error with trajectory visibility for humans (red) and SNNs (blue).

Finally, we show our system outperforms human prediction on this task.

Previous studies have shown that it is possible to make predictions from the outputs of SNNs receiving spikes generated by an event-based camera. Some of them use liquid state machines (LSM), but their predictions remain restricted to short durations (i.e., typically a few ms) (Burgsteiner et al., 2007; Kaiser et al., 2017). Another study used delays in anisotropic lateral connections (Kaplan et al., 2013). Others use a learning rule to anticipate inputs on longer prediction times but with simpler and repetitive input stimuli (Gibson et al., 2014). Our system permits predictions on longer duration by extracting motion features, as we used ball's trajectories restricted by physics' laws.

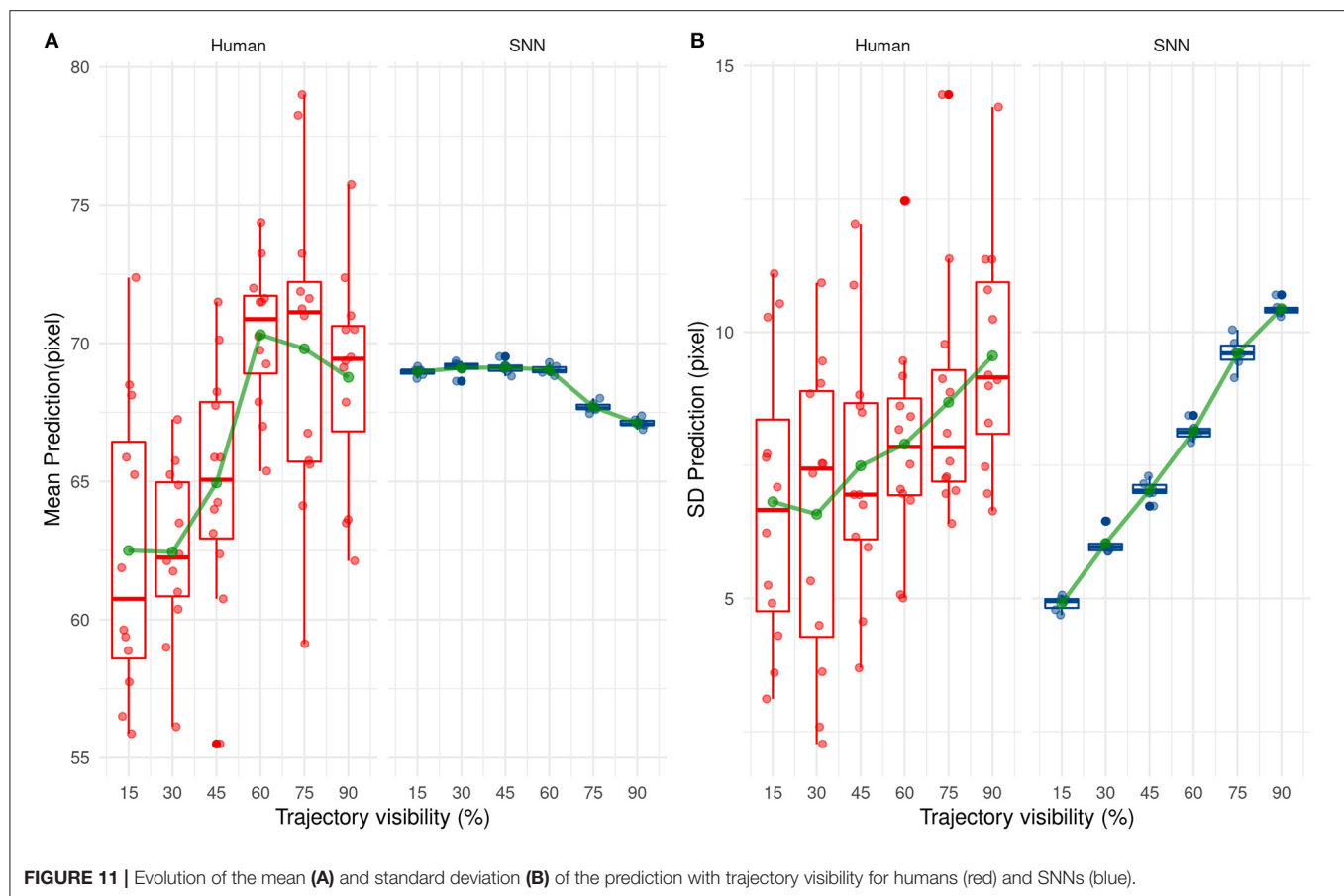
Event-based sensors allow performing sparse coding on dynamic visual scenes. In the context of this study, only a small part of the visual scene is relevant to be extracted as we only want to process motion (ball and arm). Using the testing dataset presented as the SNN's input, an average of about 9,000 spikes per second were generated by the neuroSoc. This represents only 0.26 percent of pixels for each frame (37 ± 15.5 ON/OFF spikes per frame), and shows that, in comparison to a full-frame synchronous camera, the output of the NeuroSoC was extremely sparse. This, in turn, makes the system highly

energy-efficient if embedded in a neuromorphic chip as the power consumption of spiking neural networks is determined by the number of spikes processed (Farabet et al., 2012). Through STDP, the network can then learn from these repeated sparse spatio-temporal stimuli and is thus highly capable of processing motion.

Even though our approach has been evaluated on a rather easy task, with little variations in trajectories and minimal background motion, this evaluation is still relevant to numerous situations such as basketball free throws, or objects moving along a specific constraint (e.g., cars moving on a road or pedestrian crossing a sidewalk). In a more complex situation with background motion, the motion tracking ability of the SNN could be used to discriminate the ball's motion from other objects.

The next step would be to evaluate our system on more complex trajectories (rebounds, collisions etc.) or scenes which involve unconstrained motion trajectories such as pedestrians moving along a footpath or players moving across a football field. This type of solution could also be useful in robotics (e.g., aerial drones) where real-time, energy-efficient processing is highly desirable.

In the near future, one of our aims is to embed this SNN on a chip such as the FPGA of the neuroSoc camera, allowing



us to have the acquisition sensor, the spike extraction, and the processing (SNN) in a single, low-powered, and real-time chip. Hence, this work is the first step toward showing the reliability of a simple SNN to extract relevant spatiotemporal features using the neuroSoc camera.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

ETHICS STATEMENT

The studies involving human participants were reviewed and approved by University of Toulouse local ethic committee. The patients/participants provided their written informed consent to participate in this study.

REFERENCES

Abderrahmane, N., and Miramond, B. (2019). "Information coding and hardware architecture of spiking neural networks," in *2019 22nd Euromicro Conference on Digital System Design (DSD)* (Kallithea), 291–298. doi: 10.1109/DSD.2019.00050

AUTHOR CONTRIBUTIONS

RB, BC, and TM designed the project. MP developed the camera. GD adapted a code developed by TC to run the simulations, analyzed the data, and wrote the first draft of the article. All the authors provided comments on this manuscript.

FUNDING

This research was funded by Agence National de la Recherche awarded to RB, BC, and TM, Beating Roger Federer ANR-16-CE28-0017-01.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: <https://www.frontiersin.org/articles/10.3389/fncom.2021.658764/full#supplementary-material>

Adams, S. V., and Harris, C. M. (2014). A proto-architecture for innate directionally selective visual maps. *PLoS ONE*. 9:e102908. doi: 10.1371/journal.pone.0102908

Admin (2020). *Akida Neural Processor IP*. BrainChip. Disponible sur: <https://brainchipinc.com/akida-neural-processor-ip/> (consulté le avr. 23, 2021).

- Aglioti, S. M., Cesari, P., Romani, M., and Urgesi, C. (2008). Action anticipation and motor resonance in elite basketball players. *Nat. Neurosci.* 11, 1109–1116. doi: 10.1038/nn.2182
- Barrios-Avilés, J., Iakymchuk, T., Samaniego, J., Medus, L. D., and Rosado-Muñoz, A. (2018). Movement detection with event-based cameras: comparison with frame-based cameras in robot object tracking using powerlink communication. *Electronics* 7:304. doi: 10.3390/electronics7110304
- Bi, G., and Poo, M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annu. Rev. Neurosci.* 24, 139–166. doi: 10.1146/annurev.neuro.24.1.139
- Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J. P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Netw.* 32, 339–348. doi: 10.1016/j.neunet.2012.02.022
- Brandli, C., Berner, R., Yang, M., Liu, S. C., and Delbruck, T. (2014). A 240 × 180 130 dB 3 μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circ.* 49, 2333–2341. doi: 10.1109/JSSC.2014.2342715
- Burgsteiner, H., Kröll, M., Leopold, A., and Steinbauer, G. (2007). Movement prediction from real-world images using a liquid state machine. *Appl. Intell.* 26, 99–109. doi: 10.1007/s10489-006-0007-1
- Caiman Camera. (2021). *Yumain*. Disponible sur: <https://yumain.fr/en/products/caiman-camera/> (consulté le avr. 22, 2021).
- Caporale, N., and Dan, Y. (2008). Spike timing-dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.* 31, 25–46. doi: 10.1146/annurev.neuro.31.060407.125639
- Chauhan, T., Masquelier, T., Montlibert, A., and Cottureau, B. R. (2018). Emergence of binocular disparity selectivity through hebbian learning. *J. Neurosci.* 38, 9563–9578. doi: 10.1523/JNEUROSCI.1259-18.2018
- Delorme, A., Perrinet, L., and Thorpe, S. J. (2001). Networks of integrate-and-fire neurons using Rank Order Coding B: Spike timing dependent plasticity and emergence of orientation selectivity. *Neurocomputing* 38–40, 539–545. doi: 10.1016/S0925-2312(01)00403-9
- 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
- Eurich, C. W., Pawelzik, K., Ernst, U., Thiel, A., Cowan, J. D., and Milton, J. G. (2000). Delay adaptation in the nervous system. *Neurocomputing* 32–33, 741–748. doi: 10.1016/S0925-2312(00)00239-3
- Farabet, C., Paz, R., Pérez-Carrasco, J., Zamarreño-Ramos, 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
- Farrow, D., and Abernethy, B. (2003). Do expertise and the degree of perception-action coupling affect natural anticipatory performance? *Perception* 32, 1127–1139. doi: 10.1068/p3323
- Faul, F., Erdfelder, E., Lang, A. G., and Buchner, A. (2007). G*Power 3: a flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behav. Res. Methods* 39, 175–191. doi: 10.3758/BF03193146
- Gibson, T. A., Henderson, J. A., and Wiles, J. (2014). “Predicting temporal sequences using an event-based spiking neural network incorporating learnable delays,” in *2014 International Joint Conference on Neural Networks (IJCNN)* (Beijing), 3213–3220. doi: 10.1109/IJCNN.2014.6889850
- Iyer, L. R., Chua, Y., and Li, H. (2021). Is neuromorphic MNIST neuromorphic? Analyzing the discriminative power of neuromorphic datasets in the time domain. *Front. Neurosci.* 15:608567. doi: 10.3389/fnins.2021.608567
- Kaiser, J., Stal, R., Subramoney, A., Roennau, A., and Dillmann, R. (2017). Scaling up liquid state machines to predict over address events from dynamic vision sensors. *Bioinspir. Biomim.* 12:055001. doi: 10.1088/1748-3190/aa7663
- Kaplan, B. A., Lansner, A., Masson, G. S., and Perrinet, L. U. (2013). Anisotropic connectivity implements motion-based prediction in a spiking neural network. *Front. Comput. Neurosci.* 7:112. doi: 10.3389/fncom.2013.00112
- 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
- Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435
- Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508
- Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128\$ ||times\$ 128 120 dB 15μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circ.* 43, 566–576. doi: 10.1109/JSSC.2007.914337
- Liu, S. C., van Schaik, A., Minch, B. A., and Delbruck, T. (2014). Asynchronous binaural spatial audition sensor with 2\$ || times\$, \$64\$ ||, \$4 channel output. *IEEE Trans. Biomed. Circ. Syst.* 8, 453–464. doi: 10.1109/TBCAS.2013.2281834
- Masquelier, T. (2012). Relative spike time coding and STDP-based orientation selectivity in the early visual system in natural continuous and saccadic vision: a computational model. *J. Comput. Neurosci.* 32, 425–441. doi: 10.1007/s10827-011-0361-9
- Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2008). Spike Timing dependent plasticity finds the start of repeating patterns in continuous spike trains. *PLoS ONE* 3:e1377. doi: 10.1371/journal.pone.0001377
- Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Computat. Biol.* 3:e31. doi: 10.1371/journal.pcbi.0030031
- Merriault, P., Dupuis, Y., Boutteau, R., Vasseur, P., and Savatier, X. (2017). A study of vicon system positioning performance. *Sensors* 17:1591. doi: 10.3390/s17071591
- Mohammadi, A., Asadi, H., Mohamed, S., Nelson, K., and Nahavandi, S. (2017). “OpenGA, a C++ genetic algorithm library, in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (Banff, AB), 2051–2056. doi: 10.1109/SMC.2017.8122921
- Moraitis, T., Sebastian, A., and Eleftheriou, E. (2020). Short-term synaptic plasticity optimally models continuous environments. *arXiv 2009.06808 [cs, q-bio]*. Disponible sur: <http://arxiv.org/abs/2009.06808> (consulté le avr. 23, 2021).
- Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 6178–6190. doi: 10.1109/TNNLS.2018.2826721
- Orchard, G., Benosman, R., Etienne-Cummings, R., and Thakor, N., V. (2013). “A spiking neural network architecture for visual motion estimation,” in *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Rotterdam), 298–301. doi: 10.1109/BioCAS.2013.6679698
- Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2017). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437
- Paredes-Valles, F., Scheper, K. Y. W., and de Croon, G. C. H. E. (2019). “Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: from events to global motion perception,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42, 2051–2064. doi: 10.1109/TPAMI.2019.2903179
- Perrinet, L., Samuelides, M., and Thorpe, S. (2004). Sparse spike coding in an asynchronous feed-forward multi-layer neural network using matching pursuit. *Neurocomputing* 57, 125–134. doi: 10.1016/j.neucom.2004.01.010
- 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. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid-State Circuits* 46, 259–275. doi: 10.1109/JSSC.2010.2085952
- Rawat, W., and Wang, Z. (2017). Deep convolutional neural networks for image classification: a comprehensive review. *Neural Comput.* 29, 2352–2449. doi: 10.1162/neco_a_00990
- 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
- Schrimpf, M., Kubilius, J., Hong, H., Majaj, N. J., Rajalingham, R., Issa, E. B., et al. (2018). Brain-score: which artificial neural network for object recognition is most brain-like?. *bioRxiv* 407007. doi: 10.1101/407007

- Son, B., Suh, Y., Kim, S., Jung, H., Kim, J. S., Shin, C., et al. (2017). "4.1 A 640×480 dynamic vision sensor with a 9μm pixel and 300Meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA), 66–67. doi: 10.1109/ISSCC.2017.7870263
- SP730 Near-IR/Colorless Dichroic Block Shortpass Filter (2015). *MidOpt*. Disponible sur: <https://midopt.com/filters/sp730/> (consulté le nov. 23, 2020).
- Spike Event Sensor. (2021). *Yumain*. Disponible sur: <https://yumain.fr/en/products/s-e-s-spike-event-sensor/> (consulté le avr. 23, 2021).
- Taunayazov, T., Sng, W., See, H. H., Lim, B., Kuan, J., Ansari, A. F., et al. (2020). Event-driven visual-tactile sensing and learning for robots. *Présenté Robot. Sci. Syst.* doi: 10.15607/RSS.2020.XVI.020
- Tavanaei, A., and Maida, A. S. (2017). Bio-Inspired Spiking Convolutional Neural Network using Layer-wise Sparse Coding and STDP Learning. *arXiv:1611.03000 [cs]*. Disponible sur: <http://arxiv.org/abs/1611.03000> (consulté le avr. 12, 2021).
- Thiele, J. C., Bichler, O., and Dupret, A. (2018). Event-based, timescale invariant unsupervised online deep learning with STDP. *Front. Comput. Neurosci.* 12:46. doi: 10.3389/fncom.2018.00046
- Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1
- Tversky, T., and Miikkulainen, R. (2002). Modeling directional selectivity using self-organizing delay-adaptation maps. *Neurocomputing* 44–46, 679–684. doi: 10.1016/S0925-2312(02)00457-5
- Van Rullen, R., and Thorpe, S. J. (2001). Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural Comput.* 13, 1255–1283. doi: 10.1162/08997660152002852
- VanRullen, R., Guyonneau, R., and Thorpe, S. J. (2005). Spike times make sense. *Trends Neurosci.* 28, 1–4. doi: 10.1016/j.tins.2004.10.010

Conflict of Interest: MP is Chief Technology Officer at Yumain, which develops and commercializes the Neurosoc. Yumain has however nothing to do with the design of our study or the interpretation of the results. The SNN is currently under a patent examination process, with all the co-authors at the exception of MP listed as the inventors of this patent.

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

Copyright © 2021 Debat, Chauhan, Cottureau, Masquelier, Paindavoine and Baures. 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.



Understanding the Impact of Neural Variations and Random Connections on Inference

Yuan Zeng^{1*}, Zubayer Ibne Ferdous¹, Weixiang Zhang², Mufan Xu², Anlan Yu¹, Drew Patel¹, Valentin Post¹, Xiaochen Guo¹, Yevgeny Berdichevsky^{1,3} and Zhiyuan Yan¹

¹ Electrical and Computer Engineering Department, Lehigh University, Bethlehem, PA, United States, ² Electrical and Computer Engineering Department, Beihang University, Beijing, China, ³ Bioengineering Department, Lehigh University, Bethlehem, PA, United States

Recent research suggests that *in vitro* neural networks created from dissociated neurons may be used for computing and performing machine learning tasks. To develop a better artificial intelligent system, a hybrid bio-silicon computer is worth exploring, but its performance is still inferior to that of a silicon-based computer. One reason may be that a living neural network has many intrinsic properties, such as random network connectivity, high network sparsity, and large neural and synaptic variability. These properties may lead to new design considerations, and existing algorithms need to be adjusted for living neural network implementation. This work investigates the impact of neural variations and random connections on inference with learning algorithms. A two-layer hybrid bio-silicon platform is constructed and a five-step design method is proposed for the fast development of living neural network algorithms. Neural variations and dynamics are verified by fitting model parameters with biological experimental results. Random connections are generated under different connection probabilities to vary network sparsity. A multi-layer perceptron algorithm is tested with biological constraints on the MNIST dataset. The results show that a reasonable inference accuracy can be achieved despite the presence of neural variations and random network connections. A new adaptive pre-processing technique is proposed to ensure good learning accuracy with different living neural network sparsity.

Keywords: bio-silicon computer, biological neural network, living neural network, spiking neural network, sparse connections, weight constraint, random network, recurrent network

OPEN ACCESS

Edited by:

Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

Reviewed by:

Ivan Raikov,
Stanford University, United States
Jong-Ho Lee,
Seoul National University,
South Korea

*Correspondence:

Yuan Zeng
yuz615@lehigh.edu

Received: 01 October 2020

Accepted: 19 April 2021

Published: 07 June 2021

Citation:

Zeng Y, Ferdous ZI, Zhang W, Xu M, Yu A, Patel D, Post V, Guo X, Berdichevsky Y and Yan Z (2021) Understanding the Impact of Neural Variations and Random Connections on Inference. *Front. Comput. Neurosci.* 15:612937. doi: 10.3389/fncom.2021.612937

INTRODUCTION

Artificial neural networks (ANN) have shown great success in solving real-world problems (Hinton et al., 2012; He et al., 2016; Silver et al., 2017). Most widely used neural network algorithms run on silicon-based computers, where the resource requirement and energy consumption become a challenge when the network size grows (Sze et al., 2017). In contrast, neurons and synapses naturally process information in a more energy-efficient way as compared to transistors and wires in computers (De Salvo, 2018). To develop a better artificial intelligent system, several groups (Reger et al., 2000; DeMarse et al., 2001; Han, 2013; Ju et al., 2015) proposed to incorporate biological living neural networks into the silicon platform to design a hybrid bio-silicon computer.

By dissociating the animal cortex into individual cells, placing them on an adhesive dish, and maintaining them in physiological conditions for several weeks, living neurons in a dish make

random synaptic connections with each other and form an *in vitro* living neural network (Hasan and Berdichevsky, 2016). The *in vitro* neural cultures could respond to stimuli and be precisely controlled through microelectrode arrays (MEAs) (Thomas et al., 1972; Wang et al., 2006; Pizzi et al., 2007) or an optogenetics interface (Hong et al., 2015; Nguyen et al., 2019). Such a platform not only helps neuroscience studies but also makes it possible to use neurons as “devices” for learning and computing, which applies to image recognition (He et al., 2016), speech recognition (Hinton et al., 2012), and object detection (Zhao et al., 2019) tasks.

In an early hybrid bio-silicon design (Reger et al., 2000), neurons from the reticular formation of a lamprey brain were cultured and placed in a robot to guide its movement. The sensor data gathered from the robot were used as the input for the *in vitro* living neural network. Outputs of the living neural network were processed with silicon-based circuits and used to control the motor actuators of the robot. Experiments showed that, with the closed-loop interaction between the *in vitro* network and the robot, the robot's behavior would adapt to the sensory information. Later works expanded the application sets for hybrid bio-silicon designs, Dranias et al. (2013) constructed a hybrid platform to process image patterns and Ju et al. (2015) showed that such a network is capable of classifying organized temporal sequences similar to music. Other works in this area were reviewed by Heard et al. (2018). Instead of using *in vitro* living neural networks that are randomly connected, researchers also tried to control *in vitro* neural connectivity and construct living neural circuits that carry out logic functions (Hasan and Berdichevsky, 2016). There were attempts to use *in vivo* neural networks for learning too (Musk and Neuralink, 2019). These works provided proof of concept that a hybrid bio-silicon network can perform some learning tasks and solve real-world problems.

However, the capability of the hybrid bio-silicon network is still far from the silicon-based design. Ju et al. (2015) constructed a comparison experiment between a hybrid bio-silicon design and a silicon-based design with a similar network structure and learning algorithm. The *in vitro* living neural network is modeled by a liquid state machine (LSM) structure (Maass et al., 2002) in the silicon-based design. For a temporal pattern classification task, the hybrid design achieved 60% classification accuracy, which is 10% lower than the LSM-based silicon design, and far below what a state-of-the-art silicon-based design (e.g., long short term memory; Hochreiter and Schmidhuber, 1997) could achieve. Although this result showed that a hybrid network can perform the learning task, the reason behind the accuracy gap has not been studied. In general, none of the well-designed benchmarks used to assess ANN performance have been tested in the bio-silicon platform due to implementation complexity.

The inferior performance observed in experiments could come from experimental limitations (e.g., control or recording precision), living neural network properties (e.g., high variations, random connections), as well as the poor learning capability of the algorithm. To achieve a better hybrid bio-silicon design, it is important to separate the influences of each factor and clearly understand the bottlenecks. While prior works mentioned

above focused on implementation issues, this paper aims to study the impact of living neural network properties on prediction accuracy. Specifically, this work investigates the influence of neural variations and random connections on inference with an experimentally fitted biophysical model.

Contributions of the work are listed below: (1) This work proposes a new approach to the design of algorithms for living neural network implementation. Section “Living Neural Network Properties and Related Works” reviews related works and shows that none of the existing works had the same target as this paper, and none of the existing algorithms have been proved to be efficient for living neural networks. Since a living neural network has many unique properties that are not fully considered by previous works, rethinking the algorithm design with biological constraints is necessary. (2) A two-layer hybrid bio-silicon platform and a five-step design method are proposed for the living neural network algorithm study and introduced in section “Scope of the Study.” Characteristics of neurons in culture, including their variability, are captured in a biophysical model (section “Experiment Settings,” Experiment 1). The model is then transferred to a TensorFlow-based computational model through synapse weight and neuron threshold fitting to enable fast algorithm exploration (section “Experiment Settings,” Experiment 2). Accuracy between the biophysical and computational models are compared to validate that the model transfer does not lose fidelity (section “Experiment Settings,” Experiment 3). (3) A multi-layer perceptron algorithm (section “Algorithm”) is tested with biological constraints as a case study. The algorithm is adjusted for living neural network implementation with a new adaptive pre-processing technique (section “Experiment Settings,” Experiment 4), which helps the proposed neural network to achieve good learning accuracy for living neural networks with different sparsities. At last, neural variations are studied on the optimized model (section “Experiment Settings,” Experiment 5).

LIVING NEURAL NETWORK PROPERTIES AND RELATED WORKS

Living neural networks have many intrinsic properties that are important for algorithm designs. **Table 1** summarizes the living neural network properties and the difference between this work and prior bio-inspired algorithm designs. **Artificial neural network (ANN) algorithms**, which are based on the static numerical abstractions of the biological neural networks, have shown great potential on standard benchmark testing. Although the accuracy keeps improving for ANN designs, many important biological properties are omitted. For example, in living neural cultures, the information is coded, processed, and transferred through spikes. At a certain time, the output of a neuron can be a spike or no spike, depending on whether the membrane potential has crossed the threshold or not (**b1**). However, in most ANNs, a neuron is modeled by an activation function such as a sigmoid or a rectified linear unit (ReLU), where the output could be a floating-point value. A recently proposed binary neural network (Courbariaux et al., 2016) did

TABLE 1 | Living neural network properties and the comparison with other related works.

Biological properties\works		MLP[1]	RNN [2]	Binary NN [3]	SNN [4]	RSNN [5]	LNN (Proposed)
b1	Threshold neuron output				✓	✓	✓
b2	Fixed synapse type				✓	✓	✓
b3	Synapse strength constraint						✓
b4	Neuron and synapse dynamics				✓	✓	✓ (fitted)
b5	Neuron and synapse variability	✓ [6]	✓ [6]	✓ [6]	✓	✓	✓ (fitted)
b6	Sparse connectivity	✓ [7]	✓ [7]	✓ [7]	✓ 7	✓	✓
b7	Random connectivity					✓	✓
b8	Recurrent connectivity		✓			✓	✓
Design goals		a1	a1	a2	a2	a2	a4

The check mark C means one or more works in this category have captured the properties. [1]: LeCun et al. (1998), [2]: Hochreiter and Schmidhuber (1997), [3]: Courbariaux et al. (2016), [4]: Maass (1997), [5]: Maass et al. (2002), [6]: only synapse variation is modeled by randomly initialed weights, [7]: Han et al. (2015), [8]: Krizhevsky et al. (2012), [9]: Sacramento et al. (2018).

capture the threshold neuron output to make the algorithm more hardware efficient. However, in the binary neural network, detailed neuron dynamics such as neuron spike frequency adaptation and refractory period (Liu and Wang, 2001) (b4) are not modeled. In a living neural culture, the synaptic weights are positive or negative, depending on whether it is coming from an excitatory or inhibitory neuron, respectively (Chen and Dzakpasu, 2010) (b2). The synaptic weights are also normally constrained to a range in living neural cultures, $2 \times$ larger and $0.5 \times$ smaller than the initial weights (Bi and Poo, 1998) (b3). Both the neuron and the synapse have complicated dynamics and high variability (b5). None of the existing ANN designs consider these. In a living neural culture, the network connections are randomly formed, the probability of connection between any pair of neurons is based on the distance between them (b7). The overall connectivity of the network is typically less than 40% (Barral and Reyes, 2016) and a living neural network is very sparse (b6). Recurrent connections also exist (Amit and Brunel, 1997) (b8). Some network-level properties of a living neural network are captured by existing ANNs. For example, by utilizing recurrent connections, a network could “memorize” past content and be able to predict sequences. Another example could be pruning technology (Han et al., 2015), which cuts the unnecessary connections of a trained network to make the algorithm converge faster, as well as reduce the hardware cost. Overall, as summarized in Table 2, existing ANNs are designed for high performance and efficient hardware implementations (a1). Hence, well-designed ANN algorithms may not be efficient when running on living neural networks because many necessary constraints are omitted.

Spiking neural network (SNN) algorithms emulate spiking dynamics at different levels by using different neuron and synapse models. Because the spiking feature is captured, the network is “event-driven” rather than continuously processing.

TABLE 2 | Different directions for bio-inspired algorithm designs*.

a1	Improve the learning capability for real word tasks, e.g., ANNs (MLP [1], LSTM [2], CNN [8])
a2	Improve hardware efficiency, e.g., binaryNN [3], pruning [7], SNN [4], RSNN [5]
a3	Provide hypothesis for biophysical mechanisms, e.g., dendritic backpropagation [9]
a4	Adjust or design algorithms for living neural network implementation, e.g., LNN (proposed)

*a1–a3 normally target on silicon implementation. Citations described in the caption of Table 1.

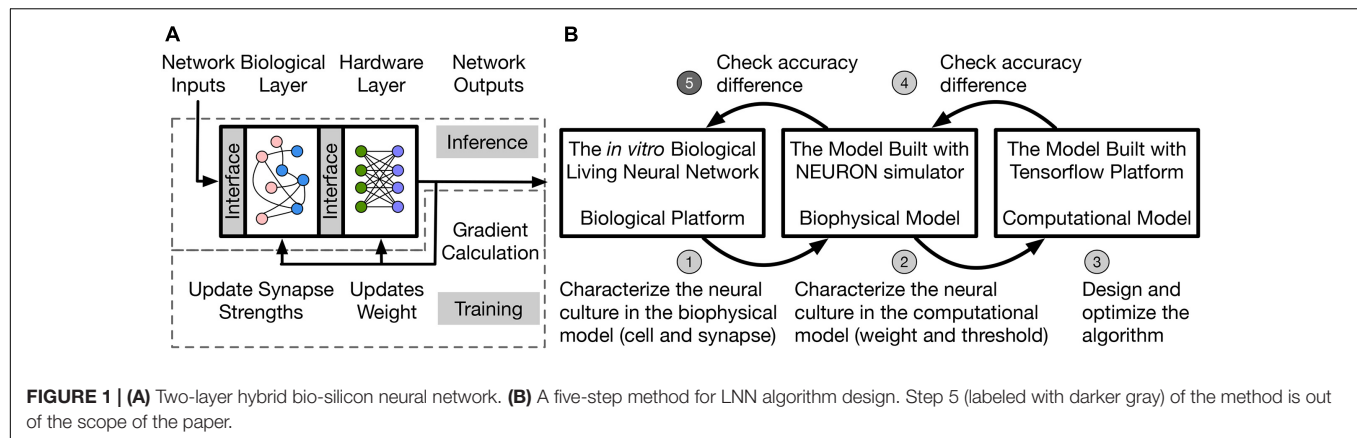
Therefore, SNNs are normally more energy efficient as compared to the ANN designs when running on hardware (a2). However, the non-continuous threshold function for SNNs also brings challenges for the training algorithm design. The powerful backpropagation algorithm for ANNs needs to be adjusted for SNNs, and many prior SNN works focused on this direction (Lee et al., 2016; Huh and Sejnowski, 2018; Wu et al., 2019). Besides the backpropagation approach mentioned above, there are explorations on spike time-dependent plasticity (STDP)-based training algorithms and unsupervised learning approaches (Diehl and Cook, 2015; Iakymchuk et al., 2015). Another group of works tried to incorporate more biological properties into the computational models to provide new hypotheses for biophysical mechanisms (a3). For example, Sacramento et al. (2018) capture more detailed neuron dynamics by modeling both the soma and dendritic compartments. This work provides a hypothesis that the dendritic micro-circuit provides a similar effect as the backpropagation algorithm in ANNs. Another example is the well-known blue brain project (Hill and Markram, 2008), which is trying to build biologically detailed digital reconstructions.

Unlike prior SNN works, where one or more properties are utilized for a unique design purpose, all living neural network properties need to be considered when targeting living neural network implementation (a4). In this work, we call algorithms to be implemented on living neural networks **living neural network (LNN) algorithms**. This work focuses on understanding the impact of neural variations and random connections to LNN algorithm design, which is an important step toward building efficient bio-silicon computers.

ALGORITHM STUDY METHOD

Scope of the Study

A two-layer hybrid bio-silicon neural network (Figure 1A) is targeted for algorithm study. The first layer is to be implemented in an *in vitro* living neural network and named **biological layer** in this paper. The second layer is computational and can run on general purpose computers or accelerators, and is named the **hardware layer** in this paper. More details about the network structure and design choice will be introduced in section “Network Structure and Data Representation.” To improve the algorithm exploration speed without losing fidelity, the *in vitro* biological living neural network (**biological platform** in Figure 1B) is represented by two different models in this work.



One is a **biophysical model**, which uses the NEURON simulator (Hines and Carnevale, 2001) to implement the biological layer. In this model, neurons are represented with the two-compartment Pinsky-Rinzel model (Pinsky and Rinzel, 1994), and synapses are represented with the alpha function model (Sterratt et al., 2011a). To further speed up the simulation, a **computational model** built with TensorFlow (Abadi et al., 2016) is used to model the living neural network, which uses the threshold function as the neuron activation function and models synapse as a floating-point value without dynamics.

In this work, the living neural network properties are converted into a simple computational model for fast algorithm design and optimization through a five-step method: (1) Biological experiments are conducted to determine neural and synaptic parameters; (2) the biophysical model captures the living neural network properties and variability by neuron and synapse parameter fitting from experimental data. However, the simulation speed for the biophysical layer is slow because of the large number of differential equations that are involved in modeling ion channels and synapses. Therefore, a simplified computational model with fast simulation speed is used and the living neural network properties are transferred into this model by fitting the weight and threshold distributions; (3) the learning algorithm is designed and optimized in the computational model with fitted biological layer parameters; (4)–(5) accuracy of the algorithm is checked on the biophysical model and the living neural network. Design details of steps (1)–(4) are introduced in the rest of the paper and step (5) remains as future work. This paper studies disinhibited networks and focuses on testing the influence of realistic biological properties on the inference process. Limitations and future steps of the work are discussed in section “Discussion.”

Network Structure and Data Representation

A living neural network is randomly connected, which means other than the input-output connections, connections also exist between inputs, between outputs, and from outputs to inputs. As a result, the network has poly-synaptic (secondary) spikes triggered indirectly by the inputs, in addition to single-synaptic

(primary) spikes. This paper assumes that an early “cut-off” mechanism for spike counting can be applied in experiments to distinguish the primary spikes from the secondary spikes, because the primary spikes normally happen before the secondary spikes. With this assumption, a living neural network can be observed as a feedforward network. In the biophysical and computational model, the biological layer is modeled with a 40% connectivity (Barral and Reyes, 2016). The hardware layer is fully connected. This work uses the MNIST dataset to evaluate network performance. Each MNIST image has 28×28 pixels in grayscale. Since controlling 784 input neurons is difficult, each MNIST image is compressed to 14×14 pixels. Section “Network and Algorithm Optimization Methods” describes the details of different pre-processing methods to compress the images. MNIST contains ten groups of digits, therefore, the network has 196 inputs, 10 outputs, and a varying number of hidden layer neurons.

An example of using the hybrid neural network for digit recognition is shown in **Figure 2**. After pre-processing and compression, the pixel values are turned into binary (zero or one) and used as the network inputs to the biological layer (**Figure 2A**). Each input neuron corresponds to a pixel in the image. For the biophysical model, channelrhodopsin-2 (ChR2) (Nagel et al., 2003) is simulated as a light-gated ion channel. By controlling the light intensity and the duration, current (the blue bar in **Figure 2A**) is generated to evoke a single spike for the input neuron if the corresponding pixel value is one. Detailed settings used for our study are introduced in section “Experiment Settings,” Experiment 1. For one input image, currents are generated for different input neurons at the same time. This kind of deterministic binary data representation, instead of spike train representation, is used to distinguish the primary output spikes from the secondary output spikes. Data are represented as floating-point numbers for the hardware layer (**Figure 2C**) for both biophysical and computational model.

Network activity of the biological layer is measured by observing and converting the output neuron membrane potential to binary representations through a spike detection process in the biophysical model. In this model, membrane potential changes with time. A 200 ms window is set to observe the output spiking pattern for each input image. A detection example is shown in

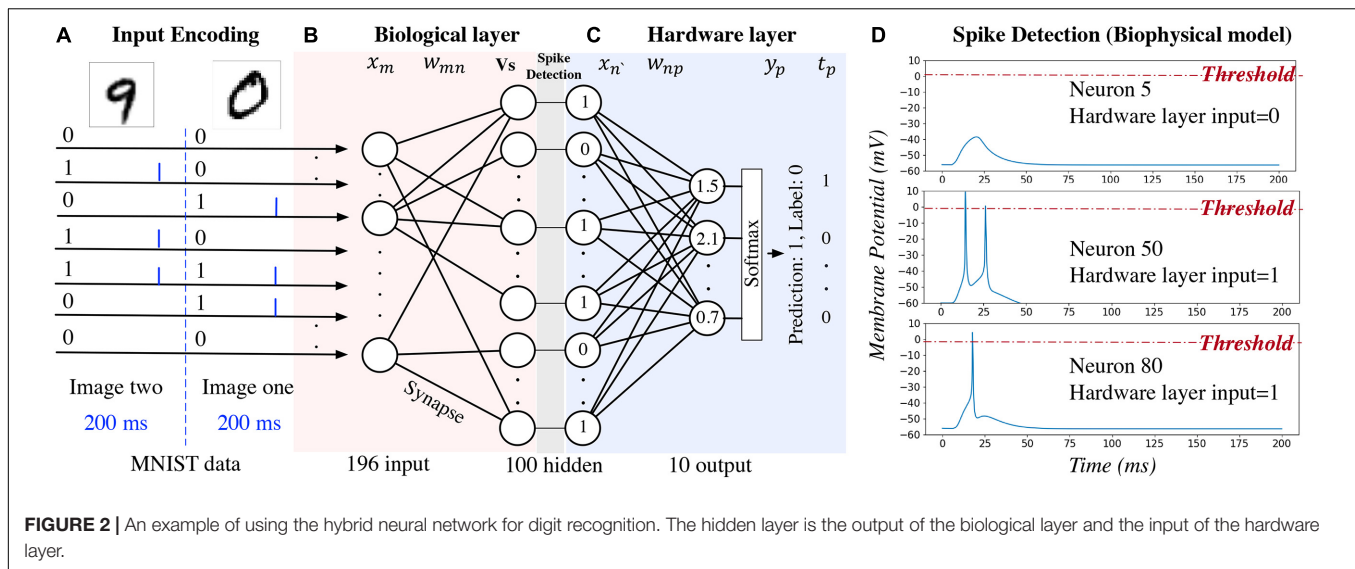


Figure 2D: membrane potential of neuron five does not reach the pre-defined threshold (0 mV); therefore, the output is zero and given as the input to the hardware layer. For neurons 50 and 80, the membrane potential exceeds the threshold, and the output value is one. For the computational model of the biological layer, the membrane potential is a floating-point value and it is converted to binary by comparing with a pre-defined threshold.

Algorithm

The algorithm study process using the biophysical and computational models is shown in Algorithm 1 with the pseudocode. Corresponding equations are described in Figure 3. In order to faithfully model a living neural network, the biological layer uses realistic parameters derived from experimental characterization of *in vitro* neurons disassociated from the cortical region of a rat brain. The hardware layer provides both high-precision data representation and flexibility for weight updates, and hence has the potential to boost performance. This work only studies the disinhibited network behavior. Disinhibited environment is easily obtained by using GABA_A receptor antagonists (Goel and Buonomano, 2013; Odawara et al., 2016) in *in vitro* experiments and it has been studied extensively in the literature. Both excitatory and inhibitory neurons exist in the network; however, synapses coming from the inhibitory neurons are prevented from functioning. Only excitatory synapses are captured for synapse parameter fitting. Correspondingly, we constrain the weights to be greater than zero in the biological layer in both biophysical and computational models to match the *in vitro* experiment setting.

To capture the neuron dynamics in the biophysical model, a two-compartment Pinsky-Rinzel neuron model (Pinsky and Rinzel, 1994) with three somatic ion channels and four dendritic ion channels is used (Figure 3) (Equations 3 and 4). An alpha synapse model (Sterratt et al., 2011a) (Equation 1) is used. The two-compartment Pinsky-Rinzel model with the alpha synapse model reproduces a variety of realistic activity patterns

in response to somatic current injection or dendritic synaptic input, which is verified by biological experiment results. Having more compartments will increase computation complexity, while

Algorithm 1: Algorithm study process for biophysical and computational model.

```

1 //Network initialization;
2 for each neuron and synapse in the biological layer do
3     if biophysical model then
4         Set neuron and synapse parameters to the fitted value (Table 3)
           discovered by Experiment 1*;
5     else if computational model then
6         Set initial weight and threshold distribution to the fitted value
           discovered by Experiment 2*
7 ;
8 for each neuron and synapse in the hardware layer do
9     Initial weight and threshold distribution are hyper-parameters, optimized
       by Experiment 3*;
10 //Learning process;
11 for each image in the training set do
12     //Inference for the biological layer;
13     Convert pixel values of the image to binary representation;
14     if biophysical model then
15         Model Equations (1)–(5) in Figure 3 for each neuron;
16     else if computational model then
17         Model Equations (9)–(10) in Figure 3 for each neuron;
18     //Inference for the hardware layer;
19     Make prediction based on Equation (11)
20     //Training;
21     Model Equations (13)–(18) to calculate loss and do back-propagation
       to update all the weights parameters for biological and Hardware layer;
22 //Validation process;
23 for each image in the testing set do
24     Do the inference steps as described in the learning process and get
       the prediction accuracy;
25 * Detailed experiment settings and results are described in section
   "Experiment Settings" and "Results" Results

```

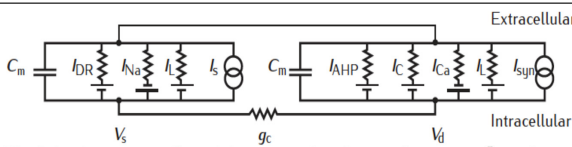
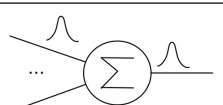
	Biophysical model	Computational model
Inference	 <p>The injection current I_s and the accumulated synaptic current I_{syn} changes the soma membrane potential V_s</p> $g_{syn}(t) = \bar{g}_{syn} \frac{t - t_s}{\tau} \exp\left(-\frac{t - t_s}{\tau}\right) \quad (1)$ $I_{syn}(t) = g_{syn}(t)(V_d(t) - E_{syn}) \quad (2)$ $C_m \frac{dV_s}{dt} = -\bar{g}_L(V_s - E_L) - g_{Na}(V_s - E_{Na}) - g_{DR}(V_s - E_k) + \frac{g_c}{p}(V_d - V_s) + \frac{I_s}{p} \quad (3)$ $C_m \frac{dV_d}{dt} = -\bar{g}_L(V_d - E_L) - g_{Ca}(V_d - E_{Ca}) + \frac{I_{syn}}{1-p} - g_{AHP}(V_d - E_k) + \frac{g_c}{1-p}(V_s - V_d) \quad (4)$ $x_{n'} = \begin{cases} 1 & V_s > V_{th,bio} \\ 0 & V_s < V_{th,bio} \end{cases} \quad (5)$	 <p>Initial:</p> $w_{mn} \sim \mathcal{N}(mean_w, sd_w) \quad (6)$ $V_{th,comp} \sim \mathcal{N}(mean_v, sd_v) \quad (7)$ $w_{mn} \geq 0 \quad (8)$ $V_s = \sum_{m=0}^M (x_m \times w_{mn}) \quad (9)$ $x_{n'} = \begin{cases} 1 & V_s > V_{th,comp} \\ 0 & V_s < V_{th,comp} \end{cases} \quad (10)$
	$y_p = softmax\left(\sum_{n=0}^N (x_{n'} \times w_{np})\right) \quad (11)$	<p>Initial:</p> $w_{np} \sim \mathcal{N}(mean_w, sd_w) \quad (12)$
Training	$Loss = -\sum_{p=0}^P t_p \ln(y_p) \quad (13)$	$w_{np} = lr \times \frac{\partial Loss}{\partial w_{np}} \quad (15)$
	$\frac{\partial Loss}{\partial w_{mn}} = lr \times (y_p - t_p) \times \left(\sum_{p=0}^P w_{np}\right) \times 1 \times x_m \quad (16)$	$w_{mn} = lr \times \frac{\partial Loss}{\partial w_{mn}} \quad (17)$
	$0.5 \times initialW_{mn} \leq W_{mn} \leq 2 \times initialW_{mn} \quad (18)$	
Terms used in this paper		
C_m	membrane capacitance	
E_L	membrane resting potential	
g_c	soma–dendrite coupling conductance	
p	the proportion of the membrane area occupied by soma	
\bar{g}_L	leakage conductance	
\bar{g}_{Na}	maximum conductance of sodium current	
\bar{g}_{DR}	maximum conductance of outward delayed-rectifier potassium current	
\bar{g}_{AHP}	maximum conductance of potassium after hyper-polarization current	
\bar{g}_{Ca}	maximum conductance of calcium current	
\bar{g}_C	maximum conductance of calcium-dependent potassium current	
\bar{g}_{syn}	maximum synaptic conductance	
τ	time constant of synaptic conductance	
E_{syn}	reversal potential of the synaptic ion channels	
t_s	time when a pre-synaptic spike arrives	
Time to first spike	time between the current injection and the maximal slope of the post synaptic membrane potential	
After-hyperpolarization voltage	The minimum voltage after the first spike and before the following spikes (if have)	

FIGURE 3 | The hybrid network learning algorithm (Martin and Jurafsky, 2009; Sterratt et al., 2011a,b).

a single-compartment model cannot capture some important neuron properties such as spike frequency adaptation behavior.

For pre-synaptic neurons, the current evoked by Chr2 (I_s) is the input to the network. For post-synaptic neurons, an

action potential is triggered when the accumulated synapse current (I_{syn}) (Equation 2) is large enough. In the biophysical model, a spike occurs when the somatic voltage (V_s) exceeds zero (Equation 5), which happens near the peak of the action

potential. For the computational model, weights and the neuron thresholds for the biological layer are initialized following a normal distribution (Equations 6 and 7), and no negative weight is allowed (Equation 8). If the sum of input-weight products at a certain neuron exceeds the threshold (Equations 9 and 10), a spike is generated. The hardware layer is fully connected with ten outputs (**Figure 2**), the weights are initialed following a normal distribution without any constraint (Equation 12). A softmax function (Martin and Jurafsky, 2009) is used to normalize the output (Equation 11). The index of the largest output is the prediction result. The cross-entropy loss (Martin and Jurafsky, 2009) is used for error backpropagation (Equation 13). The gradient of the non-differentiable hard threshold function is estimated as a constant one, which is known as the “straight-through estimator” (Bengio et al., 2013) (Equations 14–17). The weights of the biological layer are restricted to the range of $0.5 \times -2 \times$ of initial weights (Equation 18).

Experiment Settings

Experiment 1: Biophysical Model Parameter Fitting

To capture the living neural network variations in the biophysical model, parameters of the neuron and synapse models are fitted to data obtained through intracellular recordings. Nine different neurons and 12 different excitatory synapses are captured. Neural cultures were obtained by dissociating cortices of postnatal day 0 Sprague Dawley rats and plating neurons onto poly-D-lysine coated tissue culture dishes. On days *in vitro* (DIV) 12–19, neuron IV characteristics were obtained by injecting currents from -200 to 300 pA in current clamp mode with a 25 pA delta current step. Biophysical neural model (Pinsky-Rinzel) parameters were then adjusted for nine recorded cells through a multi-objective optimization approach. The defined mean square error incorporates different neuron features such as time to first spike, number of spikes, and after-hyperpolarization voltage produced by current injection. Excitatory postsynaptic currents (EPSCs) were evoked by patterned blue light stimulation of ChR2-expressing pre-synaptic neurons. The light power is set to 10mW/mm^2 and the duration is 5 ms to evoke a single spike. Synaptic parameters were then extracted by fitting an alpha function to experimentally obtained EPSC waveforms for 12 different post-synaptic neurons and the average of 10 trials are reported.

Experiment 2: Computational Model Parameter Fitting

To reduce the computational complexity, this work further converts the biophysical model into a computational neural network model with a threshold activation function. To ensure that a similar accuracy can be achieved after the conversion, this paper uses the minimum number of pre-synaptic neurons that trigger a post-synaptic neuron to fire (minPreNum) as the bridge to convert the variations in the biophysical model to the variations in the computational model. The following experiment is conducted in the biophysical model: the number of pre-synaptic neurons is varied from 1 to 20 and the pre-synaptic neurons are stimulated through injected current (I_s). The input neurons and synapses are randomly selected from

the fitted excitatory neurons and synapses, respectively. The post-synaptic neuron is sequentially selected from all the fitted neurons. A selection is allowed to repeat. The experiment is repeated 1,000 times in simulation. The outcome of this step is nine minPreNum curves for each post-synaptic neuron. The threshold and weight variations are assumed to follow a normal distribution. The expectations of each of the minPreNum curves are used to estimate the threshold variation. The nine minPreNum are aligned with peak and averaged to estimate the weight distribution. The detailed calculation process and results are shown in section “Computational Model Parameter Fitting.”

Experiment 3: Accuracy Comparison Between Biophysical and Computational Models

To validate the computational model against the biophysical model, the handwritten digit recognition task is performed with both the computational and biophysical models. In this experiment, 100 MNIST images are used, and the network size is 196-100-10 for the input-hidden-output layer. As a first step, the computational and biophysical models are compared without any variations. In this experiment, a fitted neuron and a fitted synapse are chosen from **Table 3**. The weights in the computational model are initialized to the gsyn value of the fitted synapse without variation. V_{th} in the computational model is set to match the behavior of the selected biophysical model. The network connectivity, topology, and the hardware layer weights are initialized to exactly the same for the biophysical and computational models. For the second step, both models with all variations are tested. The experiment goal is to check whether the network accuracy matches between biophysical and computational models with and without variation.

Experiment 4: Network and Algorithm Optimization

The computational model is optimized to achieve good accuracy with fitted weight and threshold distributions for the biological layer. In these experiments, 1000 MNIST images are used, and there are 196, 100, and 10 neurons in the input, hidden, and output layers, respectively. Methods for network and algorithm optimization are proposed and described in section “Network and Algorithm Optimization Methods.” Experiment results are described in section “Network and Algorithm Optimization.”

Experiment 5: Neural, Synaptic, and Network Variation Study

After network and algorithm optimization, testing accuracy is validated on the full MNIST dataset and the influence of neuron variation, synapse variation, and weight constraint are studied. These experiments are based on 60,000 training samples and 10,000 testing samples. The network has 196 inputs and 10 outputs. Hidden layer neurons vary between 100 AND 2,000.

Experiment 6: Accuracy Comparison Between Biophysical and Computational Models After Optimization

The adaptive pre-processing approach developed through computational model optimization is then validated on the

TABLE 3 | Parameters fitting results.

Cell*	Neuron parameters										Mean		
	1	2	3	4	5	6	7	8	9				
c_m ($\mu F/cm^2$)	10	8	10	15	15	10	8	10	8	10.44			
E_L (mV)	−45	−40	−45	−35	−45	−56	−50	−60	−60	−48.44			
$\bar{g}_L \times 10^{-3}$ (S/cm ²)	1.10	0.85	1.48	1.15	1.48	2.10	0.8	1.10	0.70	1.20			
$\bar{g}_{Na} \times 10^{-2}$ (S/cm ²)	8	6	15	29	25	9	7	11	8	13.11			
$\bar{g}_{DR} \times 10^{-2}$ (S/cm ²)	2	2	0.5	2	1	9.9	2	12	6	4.16			
$\bar{g}_{AHP} \times 10^{-2}$ (S/cm ²)	1	0.9	0.1	0.45	0.1	3	0.5	0.5	12	2.06			
$\bar{g}_{Ca} \times 10^{-2}$ (S/cm ²)	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.80			
$\bar{g}_C \times 10^{-2}$ (S/cm ²)	2	2	2	2.5	10	20	2	2	10	5.83			
*Cell diameter is 20 μm , p is 0.5, and g_c is 8 (S/cm ²) for all cells.													
Synapse parameters													
Cell	1	2	3	4	5	6	7	8	9	10	11	12	Mean
g_{syn} ($10^{-3} \mu S$)	0.72	0.59	0.34	0.73	0.69	0.63	1.15	0.21	1.17	2.28	0.59	0.41	0.79
τ	4.50	5.70	4.55	7.80	5.75	6.96	5.95	6.00	4.40	4.75	4.90	6.99	5.69
Delay (ms)	1.80	2.00	2.00	2.00	2.00	1.30	2.00	2.00	1.40	1.30	2.00	2.00	1.82

biophysical model. A hundred MNIST images are used for validation. The experiment settings are similar to those used in Experiment 3, except that the adaptive pre-processing optimization is applied with $Nin_b = 20$.

Network and Algorithm Optimization Methods

The image classification accuracy of the fitted computational model (the result of Experiment 2) is lower than the state-of-the-art report on the full MNIST dataset. After a closer examination of the hybrid network, **one hypothesis is that** the average percentage of firing neurons in the hidden layer (Nf_{hidden}) during the learning process directly influences the network capacity and hence the accuracy. When half of the hidden layer neurons spike on average, the network has the best learning capability. An intuitive example is that, if none of the neurons in the hidden layer spike, no matter what images are given from the dataset, the network will not learn at all. A similar situation also happens if all of the neurons in the hidden layer spike, regardless of the input image. For further analysis, considering the network structure in **Figure 2**, we introduce a parameter that is related to Nf_{hidden} :

$$f = (Sparsity \times Nin_b \times meanW) / Vth, \quad (19)$$

where *Sparsity* is calculated by the number of connected neuron pairs divided by the total number of neuron pairs between the input- and the hidden-layer neurons, Nin_b represents the number of black pixels in the input images, *meanW* represents the average weights between input- and the hidden-layer neurons for the entire dataset after training, and *Vth* is the average threshold of the hidden layer neurons. **The second hypothesis is that** when $f = 1$, the network has on average around $50\% \times N_{hidden}$ firing neurons for all of the images within the

dataset. This is because $Sparsity \times Nin_b \times meanW$ represents the expectation of x_n in Equation (5), where $x_n = \sum_{m=0}^M x_n \times w_{mn}$.

When $Sparsity \times Nin_b \times meanW = Vth$, a neuron in the hidden layer has on average 50% possibilities to fire. When f approaches zero, it is likely that none of the hidden layer neurons will fire, while when f is much larger than one, it is likely that all of the hidden layer neurons will fire. The optimization goal is to keep f close to one because the hybrid network has a greater learning capability when 50% of the hidden layer neurons are firing.

In this paper, an **adaptive pre-processing (Adpp) method** for the living neural network is proposed to shift Nf_{hidden} to 50% with a certain sparsity. In this approach, the input images are processed to achieve a target Nin_b , so that the f value is close to one. We adopt a filter-and-pool approach (LeCun et al., 1998) as the pre-processing mechanism. It is sometimes necessary to compress the input data to fit the input-bio interface of the biological layer. Fortunately, the compression can be naturally incorporated into the pre-processing. The proposed work uses 196 neurons as the inputs. To compress the 28×28 MNIST images to 14×14 , a specific filter size, stride, padding, and compression threshold need to be chosen. Relationship between these parameters are given by $compressed\ size = \frac{(original\ size - filter\ size + 2 \times padding)}{stride} + 1$. The compression threshold that turns the greyscale value to black and white after the average pooling is tuned to meet the desired Nin_b value. By using the Adpp approach, Nf_{hidden} can usually be tuned to around 50% for a typical living neural network sparsity, and thus a good accuracy can be achieved. The f value is close to one when Nf_{hidden} is close to 50%.

Besides the adaptive pre-processing method, this work studies different **gradient estimator (Est) methods** to further improve the network accuracy. Because the hidden layer of the hybrid network uses a non-continuous threshold function, the gradient

needs to be estimated. The straight-through estimator (Bengio et al., 2013), which considers the gradient as a constant one, is used for the previous experiments in this work. However, setting the gradient as one only when x_n (Figure 3) is within a small range can improve the training of the network. A set of gradient ranges around V_{th} are explored in the experiment to find the best one. Finally, the biological layer learning rate and the hardware layer parameters such as the initial weights and learning rate can be tuned as hyper-parameters.

RESULTS

Biophysical Model Parameter Fitting

Biophysical model fitting results for nine neurons and 12 synapses are shown in Table 3.

Computational Model Parameter Fitting

Figure 4A shows the nine minPreNum histograms corresponding to each post-synaptic neuron. The average of the nine minPreNum expectations is 8.2, and the standard deviation is 2.4. These values are multiplied with the average maximum synapse conductance (the average $g_{sy} = 0.0008$ in the supplementary materials is used to estimate the average weights) to derive the threshold distribution $N(0.0066, 0.0019)$. To estimate the weight distribution, the nine minPreNum are aligned with the peak and the aligned points on the curves are averaged into one curve in Figure 4B. To fit the mean and standard deviation of the curve, the minPreNum experiment is repeated using the computational model. A post-synaptic neuron with threshold = 0.0066 is used. A weight distribution of $N(0.0009, 0.0009)$ has the best fit.

Accuracy Comparison Between Biophysical and Computational Model

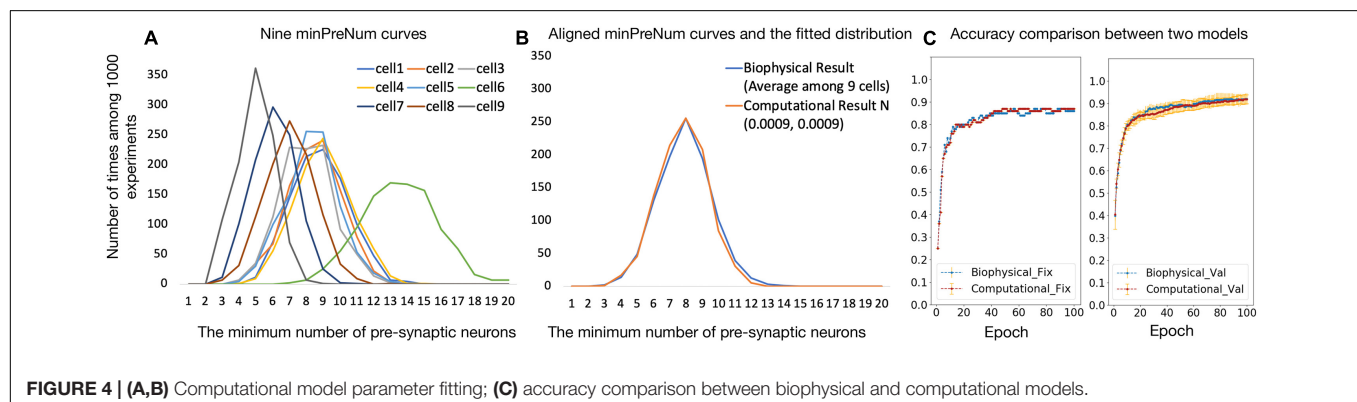
To validate the computational model against the biophysical model, the digit recognition task is performed with both the computational and the biophysical models. Experiment settings are listed in Table 4 and the testing accuracy are compared in Figure 4C, which shows that the accuracy results for the biophysical and the computational models closely match each other with and without variations.

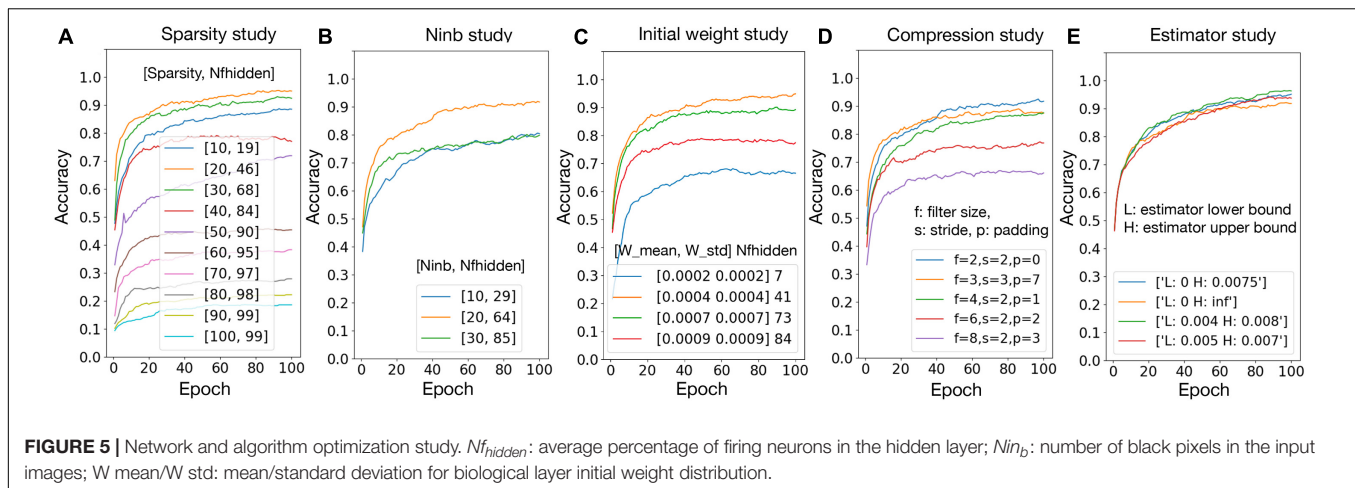
TABLE 4 | Simulation parameters.

Experiment/dataset/network	Accuracy comparison between two models/100 MNIST images, training set same as testing set/size:196-100-10
Bio layer (fix)	Sparsity: 40%, $V_{th_{cp}}: N(0.0055, 0)$, $initW_{cp}: N(0.00072, 0)$, Lr: $1e-4$
Bio layer (var)	Sparsity: 40%, $V_{th_{cp}}: N(0.0066, 0.0019)$, $initW_{cp}: N(0.0009, 0.0009)$, Lr: $1e-4$
Hardware layer	Fully connected, $initW: N(0.0009, 0.0009)$, Lr: $1e-2$
Optimization (Experiment 6)	Adpp: $Nin_b = 20$
Experiment/dataset/network	Network and algorithm optimization/1,000 MNIST images, training set same as testing set/size:196-100-10
Bio layer	Sparsity: 40%, $V_{th_{cp}}: N(0.0066, 0.0019)$, $initW: N(0.0009, 0.0009)$, Lr: $5e-6$
Hardware layer	Fully connected, $initW: N(0.0009, 0.03)$, Lr: 0.008
Optimization	Adpp: $Nin_b = 20$, Estimator range: (0, 0.0075), Adlr: bio layer initial $5e-06$, hardware layer initial 0.1, decay rate 0.1
Experiment/dataset/network	Variation study/60,000 and MNIST images for training and 10,000 for testing/size:196-(100-2,000)-10
Bio layer	Sparsity:40%, $V_{th_{cp}}: N(0.0066, 0.0019)$, $initW: N(0.0009, 0.0009)$, Lr: $5e-6$
Hardware layer	Fully connected, $initW: N(0.0009, 0.03)$, Lr:0.008
Optimization	Adpp: $Nin_b = 26$, Estimator range: (0, 0.0075), Adlr: bio layer initial $1e-05$, hardware layer initial 0.1, decay rate 0.1

Network and Algorithm Optimization

To validate the hypotheses proposed in section “Experiment Settings,” the relationship between accuracy and number of firing neurons in the hidden layer (Nf_{hidden}) is studied with the variation of network sparsity (Figure 5A), number of black pixels in the input images (Figure 5B), and the initial weights (Figure 5C), respectively. The sparsity study suggests that, when the sparsity is higher, Nf_{hidden} is larger. The best accuracy is achieved at sparsity 20%, where the corresponding Nf_{hidden} is the closest to 50% among the tested sparsity and the corresponding f values are close to one. The best accuracy is achieved at $Nin_b = 20$ and initial weight distribution $N(0.0004, 0.0004)$, where the corresponding Nf_{hidden} value is the closest to 50% among the tested parameter ranges. These observations agree with the hypotheses. Figure 5D shows the accuracy result with different combinations of the filter size, stride, and padding. The best result is given by filter size = 2, stride = 2, and padding = 0. This set of pre-processing parameters





are used in this paper. **Figure 5E** shows that passing the gradient across a neuron only when x_n falls in a smaller range can help improve the accuracy. However, for living neural networks, only the upper bound constraint of the gradient could be implemented with the “cut-off” mechanism. Therefore, (0, 0.0075) is used for the following experiments in this paper. The result of hyper-parameter tuning is listed in **Table 4**, in the network and algorithm optimization part. With all the optimization methods, a 99.5% accuracy could be achieved for the 1000 MNIST dataset.

Neural, Synaptic, and Network Variation Study

The hybrid bio-silicon neural network learning accuracy for the full MNIST dataset is reported in **Figure 6A** after adding all optimizations discussed above. For 100, 500, and 2,000 hidden layer neurons, the average testing accuracy is 85.3, 90.8, and 93.4%, respectively. For the 2,000 hidden layer neuron case, a 6% accuracy gap is observed between training and testing accuracy. When increasing the hidden layer neuron number and adding the dropout technique to alleviate the overfitting issue, a 96% testing accuracy is achieved, which is a reasonable accuracy with biological constraints.

Effects of threshold and weight variations are evaluated separately in **Figures 6B,C**, respectively. Within the tested range, with the increase of the threshold variation, network accuracy slightly increases first and then starts to decrease. This is because a larger threshold variation brings too much noise to the network, which goes beyond the ability of the algorithm. With the increase of the synaptic weight variation, the network accuracy increases first then saturates. This is because the weights are trainable, and a larger initial weight variation enlarged the synapse changing space, since the weights are constrained to $0.5 \times -2 \times$ of the initial weights. In **Figure 6D**, network accuracy is studied with the weight constraint. Synapse weights are initialized randomly but are constrained in a fixed range. Results suggest that, the larger the weight range, the higher the accuracy. Overall, a relatively good accuracy could be achieved with a realistic threshold, synapse weight, and weight constraints.

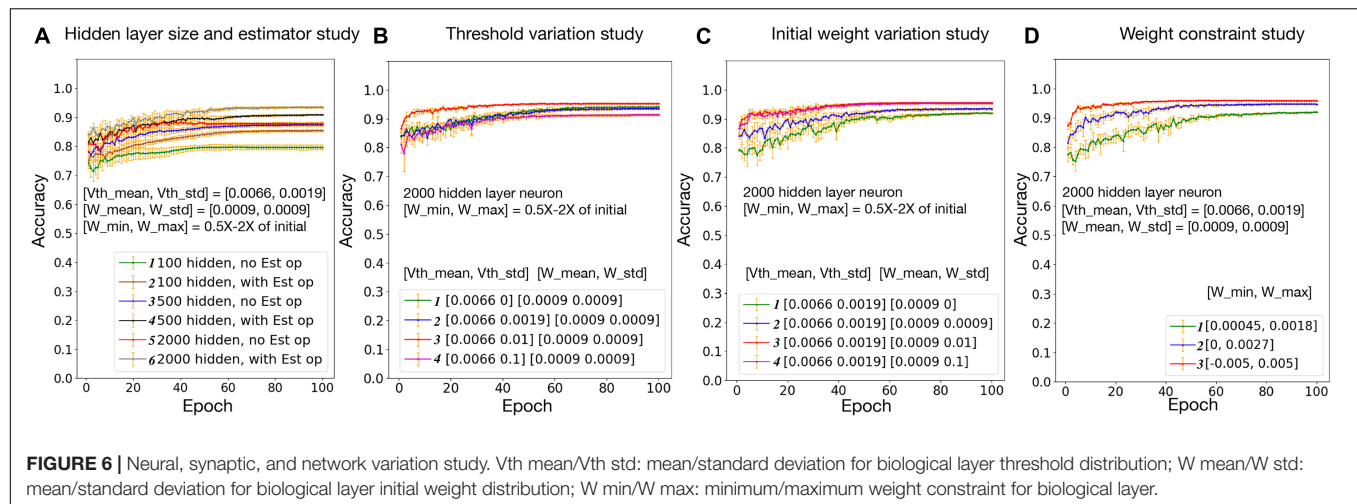
Accuracy Comparison Between Biophysical and Computational Models After Optimization

The accuracy comparison result between the biophysical and computational model before and after the network and algorithm optimization is shown in **Figure 7**. After optimization, the testing accuracy and converge speed on the 100 MNIST dataset improved for both models. The accuracy also matched well between the biophysical and computational model after optimization. This verified the effectiveness of the optimization method on the biophysical model.

DISCUSSION

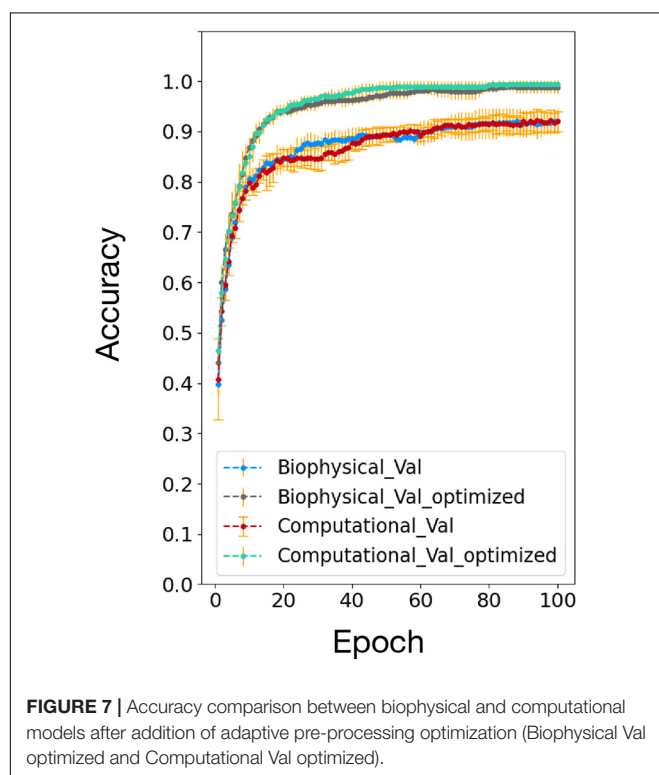
In this paper, a hybrid bio-silicon neural network is proposed and studied using both biophysical and computational models. Random network connections, as well as realistic threshold and synapse weight variations are considered. With proposed optimization methods, a 96% accuracy is achieved in simulation using a living neural network-fitted computational model. Simulation suggests that biologically plausible inference is not the major reason for a poorly performed bio-silicon computer; hence, living neurons could be used to design a learning machine.

This paper focuses on testing the influence of realistic biological properties on the inference process. Thus, the training approach is set to be the same as the conventional backpropagation algorithm to ensure a fair comparison between LNN and ANN accuracy. How to update the weights in a biologically plausible manner and ensure training efficiency is beyond the scope of this paper and will be carried out in our future work. Potential solutions could be updating weights through a supervised spike time-dependent plasticity (STDP) algorithm as shown in Zeng et al. (2018), or assigning blame by multiplying errors by random synaptic weights (Lillicrap et al., 2016). The functional equivalence between the NEURON-based biophysical model and the TensorFlow-based computation model are validated with a relatively small dataset



of 100 MNIST images, because training the NEURON model is very time-consuming and the full MNIST simulation cannot be finished within a reasonable amount of time. Only the adaptive pre-processing method is applied to validate the computational optimization on the biophysical model. For a living neural network, the network topology cannot be easily controlled. Pre-processing guarantees a good accuracy with any measured network sparsity and it is the most effective optimization approach we found through the study shown in Figure 5. With the addition of adaptive pre-processing, the network accuracy on a small dataset of 100 MNIST images reaches

100%, therefore, other optimization approaches are not applied for biophysical model validation. This work focuses on the excitatory network, which has been studied extensively in the literature and can be experimentally implemented via inhibition of GABA_A receptors (Goel and Buonomano, 2013; Odawara et al., 2016) as introduced in section “Algorithm.” With inhibitory synapse, the network activity will be more sparse and new learning features may emerge. The influence of the variability of inhibitory synapses on hybrid bio-silicon network performance is important and will be explored on the biological platform in our future work.



DATA AVAILABILITY STATEMENT

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

ETHICS STATEMENT

The animal study was reviewed and approved by Animal Care and Use Committee (IACUC) at Lehigh University.

AUTHOR CONTRIBUTIONS

YZ and XG developed the main concepts. YZ implemented the major part of the biophysical and computational models, performed the simulation, and drafted the manuscript. ZF and YB developed the biological platform. ZF performed the biological experiments and provided related data. WZ implemented the Chr2 channel in the biophysical model and helped in parameter tuning and adaptive pre-processing. MX helped in parameter tuning and adaptive learning rate. VP helped in converting biophysical variations into the computational model. All the other authors assisted in developing the concepts and writing the manuscript.

FUNDING

This work is supported by the National Science Foundation under Grant No. 1835278.

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* [preprint] arxiv.org/abs/1603.04467
- Amit, D. J., and Brunel, N. (1997). Dynamics of a recurrent network of spiking neurons before and following learning. *Network* 8, 373–404. doi: 10.1088/0954-898x_8_4_003
- Barral, J., and Reyes, A. D. (2016). Synaptic scaling rule preserves excitatory–inhibitory balance and salient neuronal network dynamics. *Nat. Neurosci.* 19:1690. doi: 10.1038/nn.4415
- Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* [preprint]
- Bi, G.-Q., and Poo, M.-M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing synaptic strength and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/jneurosci.18-24-10464.1998
- Chen, X., and Dzakpasu, R. (2010). Observed network dynamics from altering the balance between excitatory and inhibitory neurons in cultured networks. *Phys. Rev. E* 82:031907.
- 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* [preprint]
- De Salvo, B. (2018). “Brain-inspired technologies: Towards chips that think?,” in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)* (Netherlands: IEEE), 12–18.
- DeMarse, T. B., Wagenaar, D. A., Blau, A. W., and Potter, S. M. (2001). The neurally controlled animat: biological brains acting with simulated bodies. *Autonom. Rob.* 11, 305–310.
- 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
- Dranias, M. R., Ju, H., Rajaram, E., and VanDongen, A. M. (2013). Short-term memory in networks of dissociated cortical neurons. *J. Neurosci.* 33, 1940–1953. doi: 10.1523/jneurosci.2718-12.2013
- Goel, A., and Buonomano, D. V. (2013). Chronic electrical stimulation homeostatically decreases spontaneous activity, but paradoxically increases evoked network activity. *J. Neurophysiol.* 109, 1824–1836. doi: 10.1152/jn.00612.2012
- Han, J. (2013). *Computing with Simulated and Cultured Neuronal Networks*. Ph.D. thesis, Netherlands: IEEE.
- Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149* [preprint]
- Hasan, M., and Berdichevsky, Y. (2016). Neural circuits on a chip. *Micromachines* 7:157. doi: 10.3390/mi7090157
- 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* (Netherlands: IEEE) 770–778.
- Heard, M., Ford, J., Yene, N., Straiton, B., Havanas, P., and Guo, L. (2018). Advancing the neurocomputer. *Neurocomputing* 284, 36–51. doi: 10.1016/j.neucom.2018.01.021
- Hill, S., and Markram, H. (2008). “The blue brain project,” in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE)* (Netherlands: IEEE) clviii–clviii.
- Hines, M. L., and Carnevale, N. T. (2001). Neuron: a tool for neuroscientists. *Neuroscientist* 7, 123–135. doi: 10.1177/107385840100700207
- Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-R., Jaitly, N., et al. (2012). “Deep neural networks for acoustic modeling in speech recognition,” in *IEEE Signal Processing Magazine* (Netherlands: IEEE) 29.
- Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.* 9, 1735–1780.
- Hong, G., Diao, S., Antaris, A. L., and Dai, H. (2015). Carbon nanomaterials for biological imaging and nanomedicinal therapy. *Chem. Rev.* 115, 10816–10906. doi: 10.1021/acs.chemrev.5b00008
- Huh, D., and Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. *Adv. Neural Inform. Proc. Syst.* 1433–1443.
- Iakymchuk, T., Rosado-Munñoz, A., Guerrero-Martínez, J. F., Bataller-Mompeán, M., and Francés-Villora, J. V. (2015). Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. *EURASIP J. Image Video Proc.* 2015:4.
- Ju, H., Dranias, M. R., Banumurthy, G., and VanDongen, A. M. (2015). Spatiotemporal memory is an intrinsic property of networks of dissociated cortical neurons. *J. Neurosci.* 35, 4040–4051. doi: 10.1523/jneurosci.3793-14.2015
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Adv. Neural Inform. Proc. Syst.* 60, 1097–1105.
- 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
- 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, 1–10. doi: 10.1016/j.artint.2018.03.003
- 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.
- Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks* 10, 1659–1671. doi: 10.1016/s0893-6080(97)00011-7
- 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
- Martin, J. H., and Jirafsky, D. (2009). *Speech and Language Processing: an Introduction to Natural Language Processing*. New Jersey, NJ: Pearson/Prentice Hall.
- Musk, E., and Neuralink. (2019). An integrated brain-machine interface platform with thousands of channels. *J. Med. Int. Res.* 21:e16194. doi: 10.2196/16194
- Nagel, G., Szellas, T., Huhn, W., Kateriya, S., Adeishvili, N., Berthold, P., et al. (2003). Channelrhodopsin-2, a directly light-gated cation-selective membrane channel. *Proc. Natl. Acad. Sci.* 100, 13940–13945. doi: 10.1073/pnas.1936192100
- Nguyen, C., Upadhyay, H., Murphy, M., Borja, G., Rozsahegyi, E. J., Barnett, A., et al. (2019). Simultaneous voltage and calcium imaging and optogenetic stimulation with high sensitivity and a wide field of view. *Biomed. Optics Expr.* 10, 789–806. doi: 10.1364/boe.10.000789
- Odawara, A., Katoh, H., Matsuda, N., and Suzuki, I. (2016). Physiological maturation and drug responses of human induced pluripotent stem cell-derived cortical neuronal networks in long-term culture. *Scientif. Rep.* 6, 1–14.
- Pinsky, P. F., and Rinzel, J. (1994). Intrinsic and network rhythmicogenesis in a reduced traub model for ca3 neurons. *J. Comput. Neurosci.* 1, 39–60. doi: 10.1007/bf00962717
- Pizzi, R., Cino, G., Gelain, F., Rossetti, D., and Vescovi, A. (2007). Learning in human neural networks on microelectrode arrays. *Biosystems* 88, 1–15. doi: 10.1016/j.biosystems.2006.03.012
- Reger, B. D., Fleming, K. M., Sanguineti, V., Alford, S., and Mussa-Ivaldi, F. A. (2000). Connecting brains to robots: an artificial body for studying the computational properties of neural tissues. *Artificial Life* 6, 307–324. doi: 10.1162/106454600300103656
- Sacramento, J., Costa, R. P., Bengio, Y., and Senn, W. (2018). Dendritic cortical microcircuits approximate the backpropagation algorithm. *Adv. Neural Inform. Proc. Syst.* 2018, 8721–8732.

ACKNOWLEDGMENTS

This manuscript has been released as a pre-print at <https://arxiv.org/pdf/1905.11594.pdf> (Zeng et al., 2019).

- 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, 354–359. doi: 10.1038/nature24270
- Sterratt, D., Graham, B., Gillies, A., and Willshaw, D. (2011a). *Principles of Computational Modelling in Neuroscience, Section 7.2*. Cambridge: Cambridge University Press.
- Sterratt, D., Graham, B., Gillies, A., and Willshaw, D. (2011b). *Principles of Computational Modelling in Neuroscience, Section 8.1.2*. Cambridge: Cambridge University Press.
- 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
- Thomas, C. Jr., Springer, P., Loeb, G., Berwald-Netter, Y., and Okun, L. (1972). A miniature microelectrode array to monitor the bioelectric activity of cultured cells. *Exp. Cell Res.* 74, 61–66. doi: 10.1016/0014-4827(72)90481-8
- Wang, K., Fishman, H. A., Dai, H., and Harris, J. S. (2006). Neural stimulation with a carbon nanotube microelectrode array. *Nano Lett.* 6, 2043–2048. doi: 10.1021/nl061241t
- Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. (2019). Direct training for spiking neural networks: Faster, larger, better. *Proc. AAAI Conf. Artif. Intell.* 33, 1311–1318. doi: 10.1609/aaai.v33i01.33011311
- Zeng, Y., Devincintis, K., Xiao, Y., Ferdous, Z. I., Guo, X., Yan, Z., et al. (2018). “A supervised stdp-based training algorithm for living neural networks,” in *In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Netherland: IEEE), 1154–1158.
- Zeng, Y., Ferdous, Z. I., Zhang, W., Xu, M., Yu, A., Patel, D., et al. (2019). Inference with hybrid bio-hardware neural networks. *arXiv preprint arXiv:1905.11594* [preprint]
- Zhao, Z.-Q., Zheng, P., Xu, S.-T., and Wu, X. (2019). Object detection with deep learning: A review. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 3212–3232.

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 Zeng, Ferdous, Zhang, Xu, Yu, Patel, Post, Guo, Berdichevsky and Yan. 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.



Quantifying the Brain Predictivity of Artificial Neural Networks With Nonlinear Response Mapping

Aditi Anand^{1,2*}, Sanchari Sen^{2,3} and Kaushik Roy²

¹ West Lafayette Junior/Senior High School, West Lafayette, IN, United States, ² Center for Brain-Inspired Computing, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States, ³ IBM Thomas J. Watson Research Center, Yorktown Heights, NY, United States

OPEN ACCESS

Edited by:

Si Wu,
Peking University, China

Reviewed by:

Bao Ge,
Shaanxi Normal University, China
Jonathan Mapelli,
University of Modena and Reggio
Emilia, Italy

*Correspondence:

Aditi Anand
anand86@purdue.edu

Received: 24 September 2020

Accepted: 26 July 2021

Published: 24 August 2021

Citation:

Anand A, Sen S and Roy K (2021)
Quantifying the Brain Predictivity of
Artificial Neural Networks With
Nonlinear Response Mapping.
Front. Comput. Neurosci. 15:609721.
doi: 10.3389/fncom.2021.609721

Quantifying the similarity between artificial neural networks (ANNs) and their biological counterparts is an important step toward building more brain-like artificial intelligence systems. Recent efforts in this direction use *neural predictivity*, or the ability to predict the responses of a biological brain given the information in an ANN (such as its internal activations), when both are presented with the same stimulus. We propose a new approach to quantifying neural predictivity by explicitly mapping the activations of an ANN to brain responses with a non-linear function, and measuring the error between the predicted and actual brain responses. Further, we propose to use a neural network to approximate this mapping function by training it on a set of neural recordings. The proposed method was implemented within the TensorFlow framework and evaluated on a suite of 8 state-of-the-art image recognition ANNs. Our experiments suggest that the use of a non-linear mapping function leads to higher neural predictivity. Our findings also reaffirm the observation that the latest advances in classification performance of image recognition ANNs are not matched by improvements in their neural predictivity. Finally, we examine the impact of pruning, a widely used ANN optimization, on neural predictivity, and demonstrate that network sparsity leads to higher neural predictivity.

Keywords: artificial neural networks (ANN), brain-inspired computing, neuromorphic systems, brain similarity, neural recordings, neural predictivity

INTRODUCTION

The fields of machine learning and neuroscience have a long and deeply intertwined history (Hassabis et al., 2017). In the quest for developing intelligent systems capable of learning and thinking by themselves, researchers have repeatedly looked for inspirations in the biological brain. The first generation of Artificial Neural Networks (ANNs) developed in the 1950s utilized perceptrons, which are abstract mathematical models of biological neurons (Rosenblatt, 1958). In subsequent generations of ANNs, engineering efforts to successfully train these networks eventually led to the design of artificial neuron models that differ from their biological counterparts. Simultaneously, researchers continued to seek and implement biological inspirations for improving

ANNs, including their structure and function. For instance, multi-layer convolutional neural networks developed in the 1990s (Fukushima, 1980; Lecun et al., 1998) were heavily inspired by the functioning of simple and complex cells in the human visual cortex (Hubel and Weisel, 1962). More recently, the development of attention networks (Vaswani et al., 2017) was motivated by the observation that human brains “attend to” certain parts of inputs when processing large amounts of information.

While the desire to emulate more advanced functions of biological brains serves as one driver of brain-inspiration in the field of ANNs, a second, equally important motivation arises from the need for efficiency. While ANNs have matched or surpassed human performance in many machine learning tasks, including image recognition, machine translation and speech recognition, the computational cost required to do so is quite high and increasing rapidly. Amidst the justified excitement about the success of artificial intelligence in man vs. machine contests such as IBM’s Watson (IBM) and Google’s AlphaGo (Deepmind AlphaGo), the gap in energy efficiency between artificial and natural intelligence continues to grow. Improved energy efficiency is crucial in the face of exploding computational requirements for training state-of-the-art ANNs on the one hand (AI and Compute, 1998), and the need to deploy them in highly energy-constrained energy devices on the other hand (Venkataramani et al., 2016). Recent efforts also suggest that biologically inspired mechanisms have the potential to improve the robustness of ANNs to adversarial attacks (Sharmin et al., 2019; Dapello et al., 2020).

Several efforts have explored the use of biologically inspired concepts for improving the energy efficiency and robustness of ANNs, or allowing them to learn from less data. Among these efforts, one group attempts to increase *representational similarity* at the individual neuronal and synaptic level. For instance, spiking neural networks comprise of neurons mimicking the firing behavior of biological neurons while employing different neural coding schemes (Maass, 1997). A second group of efforts explore biologically inspired learning rules like Spike-Timing-Dependent Plasticity (STDP) (Bi and Poo, 1998). Finally, other efforts attempt to create ANNs with topologies that are derived from neuroanatomy (Riesenhuber and Poggio, 1999). In summary, prior efforts have taken various approaches in the attempt to identify desirable features of biological brains and embody them in ANNs.

In this work, we focus on quantifying the information similarity between ANNs and biological networks by comparing their internal responses to a given input stimulus (Schrimpf et al., 2018, 2020). This approach was pioneered by Brain-score (Schrimpf et al., 2018), which quantifies information similarity through a combination of a behavioral sub-score and a neural predictivity sub-score. We specifically focus on *neural predictivity*, which refers to the ability to predict the responses of a biological brain given the information from an ANN (such as its internal activations), when both are presented with the same stimulus. Brain-Score utilizes the Pearson correlation coefficient to capture the correlation between ANN activations and neural recordings from the macaque visual

cortex (Schrimpf et al., 2018). The use of Pearson’s correlation coefficient implicitly assumes a linear relationship between the ANN activations and neural responses. Alternative metrics such as Mutual Information can quantify correlation under non-linear relationships (Cover and Thomas, 2006). However, methods to compute Mutual Information are only known when the tensors being compared have the same rank and dimensions, which is not true when comparing ANN activations with neural recordings.

In this work, we advocate the use of an explicit, non-linear mapping function to predict neural responses from ANN activations. The rationale behind this approach is that ANN activations are themselves a product of non-linear transformations. In addition, there does not exist a one-to-one correspondence between ANN and brain layers, decreasing the likelihood that the relationship between ANN activations and neural recordings can be modeled by a linear function. A second key idea that we propose is the use of a neural network to approximate the mapping function itself. We note that this is a regression problem, where the form of the function mapping from ANN activations to neural recordings is unknown. Neural networks, which are known to be universal function approximators, have been successfully applied to many regression problems. Hence, we explore their use in our work.

Embodying the approach outlined above, we propose a new method for neural response prediction in order to quantify the informational similarity between an ANN and a set of brain recordings. The method utilizes a neural network, called the neural response predictor (NRP) network, to model the non-linear relationship between ANN activations and brain recordings. Input stimuli (in our case, images) are fed to the ANN, and the activations of its layers are extracted. These activations, along with the corresponding neural recordings (captured after presentation of the same stimuli to a primate) (Schrimpf et al., 2018), are then used to train the NRP network. The prediction error of the NRP network, termed NRP-error, is a quantitative measure of the ANN’s neural predictivity.

We implement the proposed method within the TensorFlow (Tensorflow, 2015) machine learning framework and apply it to calculate the NRP-errors of 8 state-of-the-art image classification ANNs. We utilize neural recordings from the IT (168 recording sites) and V4 (88 recording sites) regions of primate brains for 3,200 images (Schrimpf et al., 2018) to evaluate the proposed method. Our results demonstrate considerable improvement in neural predictivity over linear models which are used in previous approaches (Schrimpf et al., 2018). Our results reaffirm the finding that recent advances in image classification ANNs (from AlexNet to Xception) are not accompanied by an improvement in neural predictivity. Finally, we also evaluate the impact of commonly used network optimizations such as pruning on neural predictivity.

MATERIALS AND METHODS

In this section, we first describe the general concept of quantifying brain similarity through neural predictivity. We next present the proposed method to quantify neural predictivity and

finally discuss the experimental setup and methodology used to evaluate our proposal.

Quantifying Brain Similarity Through Neural Predictivity

Neural predictivity refers to the ability to predict biological neural behavior using the information inside an ANN. As illustrated in Eq. 1, one way to quantify neural predictivity is to explicitly formulate a function $f()$ that maps ANN activations into predicted neural recordings. In this equation, Act_i refers to the activations of layer i in the ANN and NR_{pred} refers to the predicted neural responses.

$$NR_{pred} = f\left(\bigcup_{i=1}^L Act_i\right) \quad (1)$$

$$NRP\text{-}error = \delta\left(NR_{pred}, NR_{measured}\right) \quad (2)$$

The inputs to the function $f()$ are the collection of activations from all or a subset of the layers of the ANN. Next, the predicted neural responses are compared to the measured neural recordings using a distance metric δ such as mean absolute error, to quantify neural response prediction error (*NRP-error*), as illustrated in Eq. 2. The NRP-error may be calculated separately for different brain sub-regions (e.g., V1, V4, and IT of the visual cortex) and then averaged to compute the overall NRP-error for the ANN. While there exists a wide range of possibilities for function $f()$, based on the fact that neural networks are universal function approximators, we propose to use a neural network to map from ANN activations to predicted neural recordings.

Neural Response Prediction Method

Our work proposes a new method for quantifying the neural predictivity of an ANN that is based on the overall approach proposed in section “Quantifying Brain Similarity Through Neural Predictivity.” The first key idea we propose is to explicitly map ANN activations into predicted neural recordings. A non-linear function is used for this mapping in order to overcome the limitations of previous work (Schrimpf et al., 2018). A second key idea is to use a neural network to approximate this non-linear mapping from ANN activations to predicted neural responses.

Figures 1A,B present the proposed method to quantify the neural predictivity for a given ANN, and given a set of neural recordings. The method consists of the following steps:

Add NRP Network to Decode ANN Activations

The NRP network is an auxiliary structure that decodes the ANN's activations into neural response predictions that can be directly compared to neural recordings in order to compute brain similarity. The structure of the NRP network is detailed in **Figure 1B**. First, activations (layer outputs) from selected layers of the ANN are passed through a layer of neurons that we call NRP-L1. Thus, the size of the input to the NRP network is defined by the number of activations in the chosen layers from the original ANN. The layer NRP-L1 has locally dense connectivity, i.e., the activations from each layer of the ANN are processed separately. This decision was made in order to keep the number

of parameters in NRP-L1 and the overall NRP network small. We then concatenate the outputs of NRP-L1 and pass them through a dense layer (NRP-L2). To enable the NRP network to model non-linear relationships, we add ReLU layers at the end of NRP-L1 and NRP-L2. The final layer in the NRP network (NRP-out) produces the predicted neural recordings. Therefore, the number of outputs of the NRP-out layer is set to be equal to the number of neural recording sites for which data is available. We evaluated the use of additional layers in the NRP network, but our experiments suggest that they do not provide improved accuracy. Overall, the NRP network forms a regression network that maps ANN activations into predicted neural responses, specifically the firing rates of the neurons at the recording sites.

Train the NRP Network

The composite network (the original ANN with added NRP layers) is trained while locking down the original ANN's weights. The training data for this composite network consists of stimuli (images) along with corresponding neural recordings from the visual cortex when the primate was presented with these stimuli. The loss function for this training is the mean squared error between the actual and predicted neural recordings. Standard gradient-based optimizers are used for this step [in our experiments, the Adam optimizer (Kingma and Ba, 2014) was found to give the best results]. A held-out set of data is used to validate the NRP network.

Network Architecture Search for the NRP Network

A key challenge faced by the proposed method arises from the limited number of neural recordings, which translates to limited training data for the NRP network. Although it is reasonable to expect this limitation to be gradually relaxed as additional experiments are performed, it is nevertheless one that must be considered in our effort. Thus, it becomes extremely important to determine an optimized configuration for the NRP network so that it has sufficient modeling capacity to predict the neural recordings, but can also be trained with the limited training data available. We address this challenge by performing a network architecture search (Elsken et al., 2019) on the NRP network. Specifically, we performed a grid search on the following hyperparameters for the NRP network: (i) ANN layers used as input to the NRP network, (ii) sizes of the NRP network layers (except NRP-out, whose outputs must match the number of neural recording sites), and (iii) learning rate.

We would like to underscore that the NRP network is not simply a part of the original ANN (e.g., more layers added to it). Instead, it should be viewed as a decoder that maps from ANN activations to a representation that can be directly compared with neural recordings. This overcomes the limitation of previous methods in scenarios where ANN representations and brain representations are not linearly related, and thus correlation metrics that assume a linear relationship are not able to accurately quantify the similarity.

Experimental Setup

The proposed method to compute the neural predictivity of an ANN was implemented using the Tensorflow

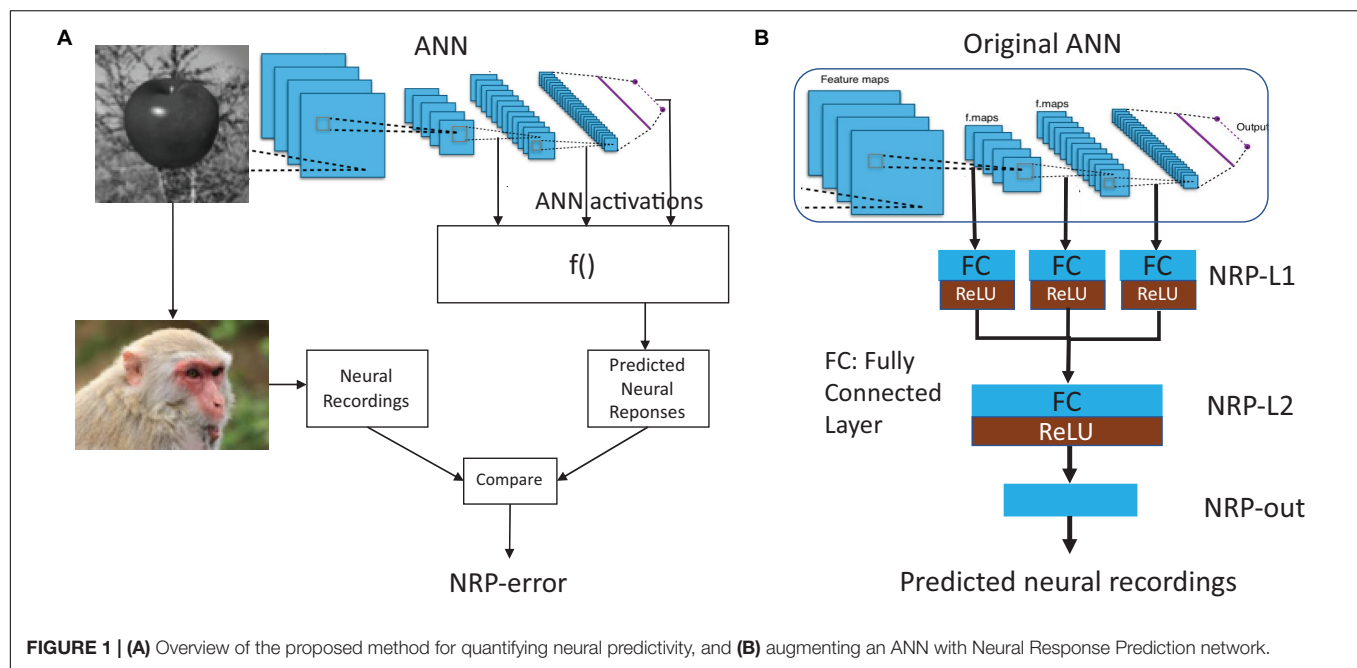


FIGURE 1 | (A) Overview of the proposed method for quantifying neural predictivity, and **(B)** augmenting an ANN with Neural Response Prediction network.

(Tensorflow, 2015) machine learning framework. NRP-errors were calculated for 8 popular image recognition ANNs that have been proposed in recent years for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015). The characteristics of these networks are described in **Table 1**.

The dataset used to train the NRP network and compute NRP-error consists of recordings from 168 neurons in the IT sub-region and 88 neurons in the V4 sub-region of the primate visual cortex (Schrumpf et al., 2018). These responses were measured when visual stimuli (3,200 images) were presented to the primates (*Rhesus macaques*) for 100 ms each immediately before these measurements were made (Schrumpf et al., 2018). Specifically, these neural recordings consist of the average neuronal firing rate for each neuron between 70 and 170 ms after the image was presented. Neuronal firing rates were normalized to the firing rates resulting from a blank gray stimulus. Note that the proposed method is generic and can be applied to recordings from additional sites or brain regions as such recordings become available. The NRP network for each ANN takes as input selected layer activations from the ANN, and produces as output

predicted firing rates for each of the recording sites. The NRP-error is the mean absolute error between the predicted firing rates and neural measurements.

NRP-errors were calculated separately for the V4 and IT regions of the visual cortex. In addition to the non-linear model used to generate predicted neural recordings for the calculation of NRP-error, we also implemented a linear regression model to predict neural recordings as a representative of previous efforts.

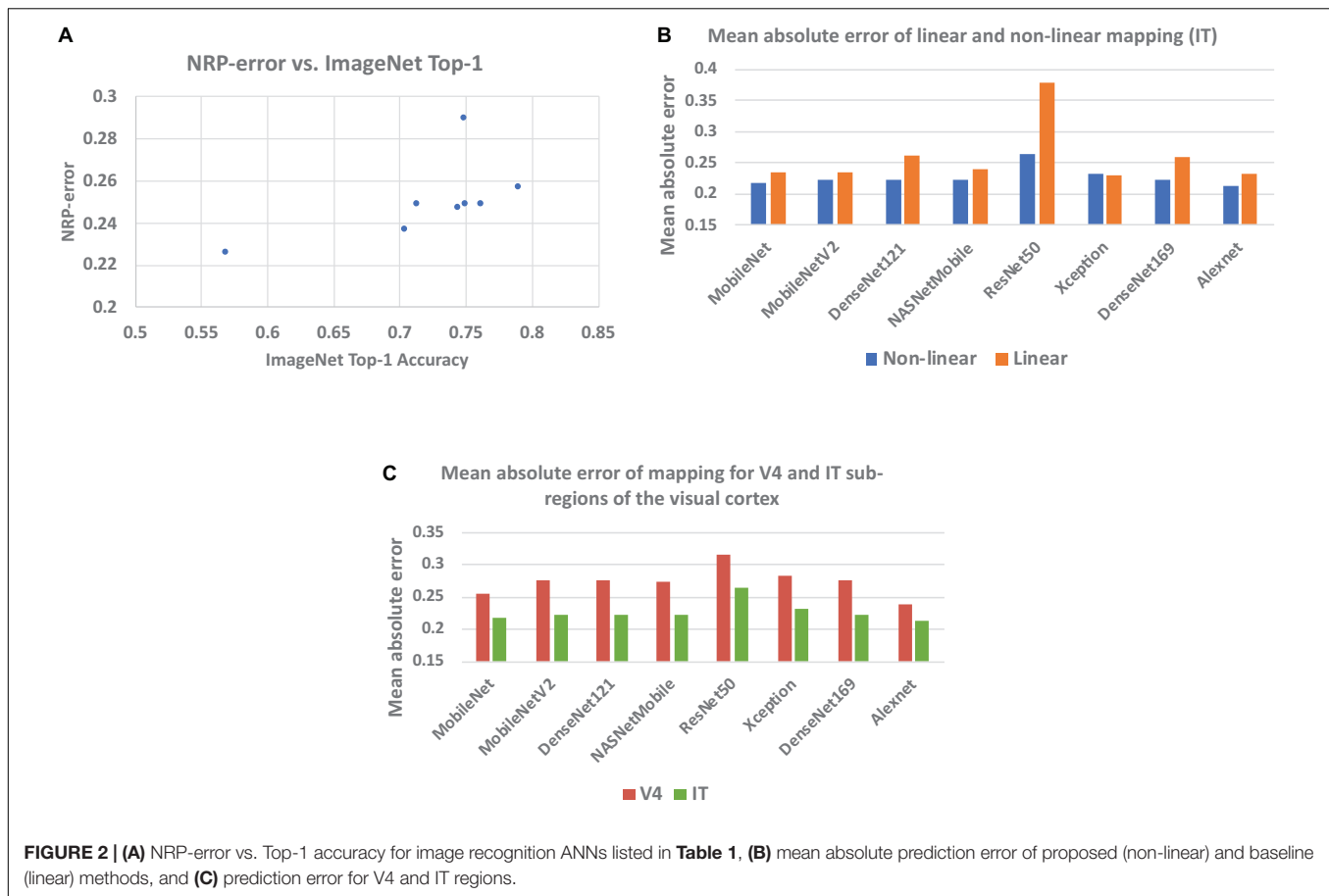
RESULTS

In this section, we discuss the results of implementing the proposed method to quantify neural predictivity of ANNs.

Table 1 presents the NRP-errors of eight different ImageNet classification ANNs. These NRP-errors were computed as the averages of the errors on the V4 and IT regions. As can be seen from the table, some of the more recent ANNs such as ResNet50 (NRP-error of 0.290) are associated with NRP-errors that are higher than older networks such as AlexNet (NRP-errors of 0.226). In fact, AlexNet achieved the lowest NRP-error, while having the lowest Top-1 accuracy, among all networks evaluated. In other words, improvements in application performance (Top-1 accuracy) have not been accompanied by increases in neural predictivity. Another observation is that deeper networks do not necessarily lead to higher neural predictivity. For example, comparing DenseNet121 and DenseNet169, we can see that the additional layers improve Top-1 accuracy but not the neural predictivity. This overall trend, illustrated in **Figure 2A**, is consistent with observations from recent efforts on quantifying brain similarity (Schrumpf et al., 2018). This is perhaps because, deeper ANNs have enabled improvements in accuracy, but have done so by adopting internal representations that are beyond and less like those used in biological systems.

TABLE 1 | Accuracies and NRP-errors of image recognition ANNs.

Network	Parameters	Top-5 accuracy	Top-1 accuracy	NRP-error
MobileNet	4,253,864	0.895	0.704	0.237
MobileNetV2	3,538,984	0.901	0.713	0.249
NASNetMobile	5,326,716	0.919	0.744	0.247
ResNet50	25,636,712	0.921	0.749	0.290
Xception	22,910,480	0.945	0.79	0.257
DenseNet121	8,062,504	0.923	0.75	0.249
DenseNet169	14,307,880	0.932	0.762	0.249
AlexNet	60,954,656	0.803	0.57	0.226



Necessity of Non-linear Mapping Function

A key feature of our work is the use of a non-linear mapping function (approximated by a neural network) to map ANN activations to predicted neural recordings in the calculation of NRP-error. This is in contrast to prior efforts, which use the Pearson correlation coefficient, effectively assuming a linear relationship between ANN activations and neural responses. In order to demonstrate the necessity of a non-linear mapping function, we also implemented a linear regression model to predict neural recordings from ANN layer activations. **Figure 2B** compares the mean absolute errors obtained from the proposed method as well as the linear regression model for the IT region. As can be seen from **Figure 2B**, our results show that a non-linear mapping function from ANN activations to predicted neural recordings significantly decreases the error of neural prediction and can hence be considered a superior predictor of an ANN's neural predictivity. The results for V4 also lead to the same conclusion. For example, in the case of ResNet-50, the mean absolute error of the linear and non-linear models are 0.379 and 0.265, respectively. This is explained by the facts that ANN layers are non-linear transformations and there is no layer-to-layer correspondence between most ANN and brain layers, making a non-linear function more suitable to model the mapping between ANN activations and neural recordings.

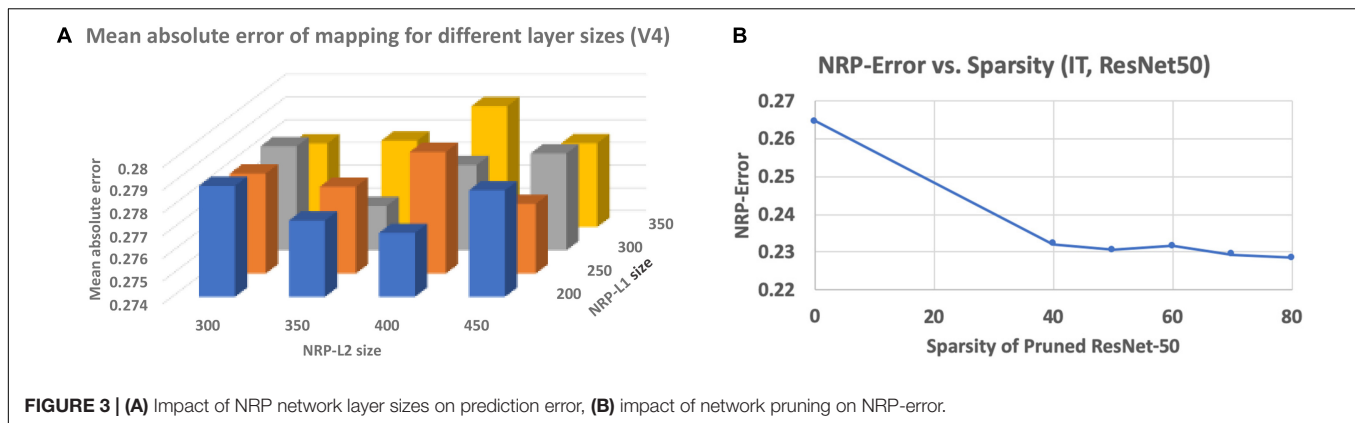
To further establish the inability of a linear model to capture the relationship between ANN activations and neural recordings, we computed the R^2 values of the linear regression models. We found the R^2 values to be less than 0.2 in all cases, which indicates that a linear model is unable to capture the relationship between ANN activations and neural recordings.

NRP-Errors for V4 and IT Sub-Regions

In order to compare the neural predictivities for the V4 and IT sub-regions of the visual cortex, NRP-errors were computed separately for both sub-regions. From the results, we observe that ANN activations predict IT neural recordings with a higher accuracy than V4 neural recordings (**Figure 2C**). This suggests that ANNs use representations that have a higher level of similarity with later visual cortex sub-regions (such as IT).

Relationship Between Neural Predictivity and Layer Sizes of NRP Network

To overcome the small amount of training data (neural recordings) available for the NRP network, a suitable configuration must be determined so that it has sufficient modeling capacity to predict the neural recordings but can also be trained with the limited training data available. In order to address this, we performed a network architecture search on the NRP network by varying the sizes and number of intermediate



layers to find a suitable configuration. Representative results obtained for the MobileNet ANN are presented in **Figure 3A**. We found that using two intermediate layers in the NRP network before the output layer is sufficient to model the mapping from ANN activations to predicted neural recordings. We also found that there is a sweet-spot of layer sizes for the NRP network that minimizes the average mean absolute error of mapping across all networks.

Impact of Network Pruning on Neural Predictivity

Finally, we investigate the impact of a popular ANN optimization technique, namely network pruning, on neural predictivity. We consider the ResNet50 ANN and applied state-of-the-art pruning algorithms (Zmora et al., 2019) to derive pruned models with varying levels of sparsity. We define sparsity as the percentage of weights that are zero-valued. We generated pruned models of the ResNet50 ANN with sparsity varying from 0 to 80%. We then applied the proposed method to the pruned models to compute the corresponding NRP-errors, and the results are presented in **Figure 3B**. The results suggest that pruning leads to a clear decrease in NRP-error, indicating a positive effect on neural predictivity. We believe this is due to the fact that pruning removes “extraneous” information from the ANN, making it easier to map its activations to the neural recordings.

DISCUSSION

Despite the rapid advances made in the field of deep learning over the past decade, biological brains still have much to teach us in the quest to build more energy-efficient and robust artificial intelligence. A key step toward drawing inspiration from biological brains is to quantify the similarity between them and their artificial counterparts. Our work takes the approach of quantifying similarity through neural predictivity, or the ability to predict neural responses from a biological brain given the internal information of ANNs. Since this is the goal of our work, we discuss closely related efforts and place our own effort in their context. We also discuss possible future directions, both in terms of improving our work and its applications.

Related Work

A recent effort that quantifies neural predictivity is Brain-Score (Schrimpf et al., 2018). Brain-Score specifically focuses on evaluating ANNs that perform core object recognition tasks, and provides a quantitative framework to compare image classification ANNs with measurements from the visual cortex of primates (firing rates for specific neurons when the primate is presented with the stimulus). It consists of a behavior sub-score and neural predictivity sub-scores for various regions of the visual cortex (V1, V2, V4, and IT). The behavior sub-score quantifies how similar the ANN’s predictions are to those made by the primate when both are presented with the same stimulus. The neural predictivity sub-scores capture how well the ANN’s activations correlate to the neural recordings from each region of the visual cortex. These sub-scores are computed as the Pearson correlation coefficient between ANN layer outputs and neural firing rates for that region.

Through the use of the Pearson correlation coefficient, Brain-Score implicitly assumes a linear relationship between ANN activations and neural firing rates. However, since ANN layers are non-linear transformations, there is no evidence to support this assumption. Moreover, there is no layer-to-layer correspondence between most ANN and brain layers, making the likelihood of a linear relationship even less likely.

Our work extends the state-of-the-art through two key ideas. First, it advocates the use of an explicit (non-linear) mapping function to predict neural responses from ANN activations in order to quantify neural predictivity. A second key idea is the use of a neural network (known to be a universal function approximator) to approximate the mapping function itself. Our experiments clearly support the merit of these proposals by demonstrating an improved ability to predict neural responses.

Future Work

One possible direction to build upon our effort would be to collect and incorporate additional neural recordings into the dataset used. A dataset with additional recording locations and more input images would allow us to train larger (and potentially more accurate) NRP networks without the risk of over-fitting. Since internal representations are greatly influenced by training, it would also be interesting to study whether networks trained

with bio-plausible learning rules (e.g., STDP) yield higher neural predictivity than ANNs trained with gradient-descent. Finally, building upon a recent result that using brain-like representations in the early layers of an ANN can lead to higher robustness, it would be interesting to study whether there is a relationship between an ANN's neural predictivity and its robustness to noise and adversarial perturbations.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

AA implemented the methods, performed the experiments, and analyzed the data. SS and KR provided mentorship and inputs

into all aspects of the research. SS assisted with experiments. All authors contributed to writing the manuscript.

FUNDING

This work was supported in part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Martin Schrimpf and James DiCarlo from the Department of Brain and Cognitive Sciences at the Massachusetts Institute of Technology for providing them with the neural recordings used in this work, and for their valuable suggestions.

REFERENCES

- AI and Compute, (2018). Available online at: <https://openai.com/blog/ai-and-compute/> (accessed August 15, 2020).
- Bi, G. Q., and Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472.
- Cover, T. M., and Thomas, J. A. (2006). *Elements of Information Theory*. Hoboken, NJ: Wiley-Interscience.
- Dapello, J., Marques, T., Schrimpf, M., Geiger, F., Cox, D. D., and DiCarlo, J. J. (2020). Simulating a primary visual cortex at the front of CNNs improves robustness to image perturbations. *bioRxiv* doi: 10.1101/2020.06.16.154542
- Deepmind AlphaGo, (2017). Available online at: <https://deepmind.com/research/case-studies/alphago-the-story-so-far> (accessed August 15, 2020).
- Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: a survey. *J. Mach. Learn. Res.* 20, 1–21.
- Fukushima, K. (1980). Neocognitron, a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernet.* 36, 193–202.
- Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron* 95, 245–258.
- Hubel, D. H., and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* 160, 106–154.
- IBM, (2011). *A Computer Called Watson*. Available online at: <https://www.ibm.com/ibm/history/ibm100/us/en/icons/watson/> (accessed August 15, 2020).
- Kingma, D. P., and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. Available online at: <https://arxiv.org/abs/1412.6980> (accessed August 15, 2020).
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324.
- 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
- Riesenhuber, M., and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nat. Neurosci.* 2, 1019–1025. doi: 10.1038/14819
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* 65, 386–408.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., and Ma, S. (2015). ImageNet large scale visual recognition challenge. *Int. J. Comp. Vis.* 115, 211–252.
- Schrimpf, M., Blank, I., Tuckute, G., Kauf, C., Hosseini, E. A., Kanwisher, N., et al. (2020). Artificial neural networks accurately predict language processing in the brain. *bioRxiv* doi: 10.1101/2020.06.26.174482
- Schrimpf, M., Kubilius, J., Hong, H., Majaj, N. J., Rajalingham, R., Issa, E. B., et al. (2018). Brain-score: which artificial neural network for object recognition is most brain-like? *bioRxiv* doi: 10.1101/407007
- Sharmin, S., Panda, P., Sarwar, S. S., Lee, C., Ponghiran, W., and Roy, K. (2019). A comprehensive analysis on adversarial robustness of spiking neural networks. *Int. Joint Conf. Neural Netw.* 63, 3493–3500.
- Tensorflow, (2015). An end-to-end open source machine learning platform. Available online at: <https://www.tensorflow.org/> (accessed August 15, 2020).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, New York, NY.
- Venkataramani, S., Roy, K., and Raghunathan, A. (2016). “Efficient embedded learning for IoT devices,” in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, 308–311.
- Zmora, N., Jacob, G., Zlotnik, L., Elharar, B., and Novik, G. (2019). Neural network distiller: a python package for DNN compression research. *arXiv*. Available online at: <https://arxiv.org/abs/1910.12232> (accessed August 15, 2020).

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.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2021 Anand, Sen 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.

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