Rectified Linear Postsynaptic Potential Function for Backpropagation in Deep Spiking Neural Networks

Malu Zhang, Jiadong Wang, Jibin Wu, Ammar Belatreche, Burin Amornpaisannon, Zhixuan Zhang, V. P. K. Miriyala, Hong Qu, Yansong Chua, Trevor E. Carlson and Haizhou Li, *Fellow, IEEE*

***Abstract*—Spiking Neural Networks (SNNs) use spatio-temporal spike patterns to represent and transmit information, which are not only biologically realistic but also suitable for ultra-low-power event-driven neuromorphic implementation. Just like other deep learning techniques, Deep Spiking Neural Networks (DeepSNNs) benefit from the deep architecture. However, the training of DeepSNNs is not straightforward because the well-studied error back-propagation (BP) algorithm is not directly applicable. In this paper, we first establish an understanding as to why error back-propagation does not work well in DeepSNNs. We then propose a simple yet efficient Rectified Linear Post-synaptic Potential function (ReL-PSP) for spiking neurons and a Spike-Timing-Dependent Back-Propagation (STDBP) learning algorithm for DeepSNNs where the timing of individual spikes is used to convey information (temporal coding), and learning (back-propagation) is performed based on spike timing in an event-driven manner. We show that DeepSNNs trained with the proposed single spike time-based learning algorithm can achieve state-of-the-art classification accuracy. Furthermore, by utilizing the trained model parameters obtained from the proposed STDBP learning algorithm, we demonstrate ultra-low-power inference operations on a recently proposed neuromorphic inference accelerator. The experimental results also show that the neuromorphic hardware consumes 0.751 mW of the total power consumption and achieves a low latency of 47.71 ms to classify an image from the MNIST dataset. Overall, this work investigates the contribution of spike timing dynamics for information encoding, synaptic plasticity and decision making, providing a new perspective to the design of future DeepSNNs and neuromorphic hardware.**

This research work is supported by Programmatic Grant No. A1687b0033 and Programmatic Grant No. I2001E0053 from the Singapore Government's Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain), and by the Science and Engineering Research Council, Agency of Science, Technology and Research, Singapore, through the National Robotics Program under Grant No. 192 25 00054. The work of J. Wu was also partially supported by the Zhejiang Lab (No.2019KC0AB02). The work of M. Zhang was aslo partially supported by the China Postdoctoral Science Foundation under Grant No.2020M680148, Zhejiang Lab's International Talent Found for Young Professionals, and National Key R&D Program of China under Grant No. 2018AAA0100202. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the supporting institutions and companies.

M. Zhang, Z. Zhang, and H. Qu are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China.

M. Zhang, J. Wang, J. Wu, Y. Chua and H. Li are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore.

B. Amornpaisannon, V. P. K. Miriyala, and T. Carlson are with the Department of Computer Science, School of Computing, National University of Singapore, Singapore.

A. Belatreche is with the Department of Computer and Information Sciences, Faculty of Engineering and Environment, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.

Corresponding author: hongqu@uestc.edu.cn

***Index Terms*—Spiking neural networks, Deep neural networks, Spike-timing-dependent learning, Event-driven, Neuromorphic hardware**

I. Introduction

THE success of Deep Neural Networks (DNNs) is attributed to their underlying deep hierarchical structure that is capable of learning representations of big data with multiple levels of abstraction [1]. Recent advances in DNNs have witnessed an increasing attention both from academia and industry with widespread applications in various areas such as image recognition [2], speech recognition [3], natural language processing [4], [5], and medical diagnosis [6]. However, training DNNs generally requires high-performance computing hardware (e.g., GPUs and computing clusters). Therefore, in power-critical computing platforms, such as edge computing, the deployment of DNNs remains drastically limited [7], [8]. Motivated by the principles of brain computing, Spiking Neural Networks (SNNs) offer a low-power alternative for neural network implementation and provide great computing potential equivalent to that of DNNs on an ultra-low-power spike-driven neuromorphic hardware [9]–[14].

However, due to the complex temporal dynamics of spiking neuronal models and the non-differentiable nature of their spiking activity, the well-known error back-propagation (BP) learning algorithm cannot not be directly applied to deep SNNs [15]. As such, SNNs have yet to match the performance of their DNN counterparts in pattern classification tasks [16]. To address the aforementioned problems, many solutions have been proposed recently [17]–[23], which can be grouped into the following three categories.

The first category includes ANN-to-SNN conversion methods where we first train an ANN, and subsequently map the pre-trained ANN weights to an SNN equivalent [24]–[34]. By carefully designing the hyperparameters of the SNN, e.g., spiking neuron model, firing threshold, we can approximate well the neural representation of the pre-trained ANN. Such ANN-to-SNN conversion benefits from the state-of-the-art ANN training algorithms. However, the conversion often leads to loss of accuracy due to approximation. A number of studies attempted to mitigate such approximation effect. For instance, the weight normalization [29]–[31] technique rescales the pre-trained weights to prevent the approximation errors from either becoming excessive or resulting in too little firing of the spiking neurons. In addition, existing studies show that training ANNs with noise-augmented data can improve the robustness and accuracy of the converted SNNs [27], [28]. Yet, despite

considerable progress in improving SNN training and their applicability, their performance is still lags behind their ANN counterparts. As the firing rate of spiking neurons is typically used to encode the activation value of artificial neurons, the spike timing information is not effectively exploited and the run-time computation is also energy-intensive when deployed on neuromorphic hardware [15], [35], [36].

The second category consists of membrane potential-driven learning algorithms, which treat the neuron's membrane potential as a differentiable signal. The discontinuities of the membrane potential at spike timings are addressed with continuous surrogate derivatives [15]. In [37]–[39], the errors are back-propagated based on the membrane potential at a single time step, which totally ignores the temporal dependency. To address this problem, SLAYER [8] and STBP [40], [41] train DeepSNNs with surrogate derivatives based on the idea of Back-propagation Through Time (BPTT) algorithm. While competitive accuracies are reported on the MNIST and CIFAR-10 datasets [41], the computational and memory requirements of BPTT-based approaches remain high because the intermediate activation values must be stored for the gradient computation.

The third category includes the spike-driven learning algorithms, which consider the timing of spikes as a relevant signal for synaptic update [36], [42]–[49]. The typical examples include SpikeProp [43] and its derivatives [44]–[47]. These algorithms rely on the assumption that the neuron membrane potential increases linearly over the infinitesimal time interval around the spike time. This allows the calculation of derivatives at each spike time and facilitates the implementation of back-propagation in multi-layer SNNs. Grounded on the time-to-first-spike (TTFS) coding, Mostafa [36] applied non-leaky integrate-and-fire neurons to avoid the problem of the non-differentiable spike function, and demonstrated competitive performance on the MNIST dataset. Along the same direction, the performance of SNNs with spike-driven learning algorithms is further improved in [48] and [49]. However, the existing spike-driven learning algorithms face several problems, for example, dead neurons and gradient exploding [36]. In [36], some constraints are imposed on the synaptic weights to overcome the dead neuron problem, and a gradient normalization strategy is used to overcome the problem of gradient exploding. These complex training strategies, however, limit the scalability of the learning algorithm.

Among the existing learning algorithms, only the spike-driven learning algorithms perform the SNNs training in a strictly event-driven manner, and are compatible with the temporal coding in which the information is carried by the timing of individual spikes that has a high level of sparsity. Hence, spike-driven learning algorithms hold great potentials to enable efficient training and inference on low-power neuromorphic devices. Recently, several neuromorphic architectures have been proposed to accelerate the SNN inference and training [24], [50]–[57]. Notably, Intel Loihi [50] can support on-chip learning with a wide range of spike-timing-dependent-plasticity (STDP) rules. During SNN inference, Loihi is claimed to be 1000$\times$ faster than the general-purpose processors such as CPUs and GPUs, while using much less power. Additionally, Srivatsa et al. [58] recently proposed a neuromorphic accelerator called You Only Spike Once (YOSO) that can leverage the sparse spiking activity to achieve ultra-low-power inferences with high classification accuracies [58].

In this work, we develop an effective spike-driven learning algorithm for training high-performance deep SNNs. This paper makes the following main contributions:

1) A comprehensive study of what makes the BP algorithm incompatible for training SNNs, including the problems of non-differentiable spike generation function, exploding gradient and dead neuron. Building on the insights from this study, we put forward a Rectified Linear Postsynaptic Potential function (ReL-PSP) for spiking neurons to resolve these problems.

2) Based on the proposed ReL-PSP, we derive a novel spike-timing-dependent BP algorithm (STDBP) for DeepSNNs. In this algorithm, the timing of spikes is used as the information carrier, and learning happens only at the spike times in a fully event-driven manner.

3) We demonstrate the effectiveness and scalability of STDBP with ReL-PSP in convolutional spiking neural networks (C-SNN), which achieves a classification accuracy of 99.5% on the MNIST dataset.

4) Lastly, using the SNN trained with the proposed STDBP learning algorithm, we demonstrate ultra-low-power and rapid inference on the recently proposed neuromorphic accelerator, YOSO.

In summary, this work not only provides novel perspectives on the significance of neuronal dynamics in information coding, synaptic plasticity, and decision making within a SNN-based computing paradigm, but also demonstrates the possibility of realizing ultra-low-power inference with high classification accuracy on the emerging neuromorphic hardware.

The remainder of this paper is organized as follows. In section II, we establish an understanding as to why error back-propagation does not work well in DeepSNNs. Section III presents a detailed description of the proposed ReL-PSP and the STDBP learning algorithm. In section IV, we introduce the recently proposed neuromorphic hardware architecture, YOSO. Section V presents a comprehensive experimental evaluation of the proposed ReL-PSP and STDBP learning algorithm. Finally, section VI discusses the results and draw conclusions.

II. Problem Analysis

Error back-propagation, specifically stochastic gradient descent, is the workhorse of learning in DNNs. However, as shown in Fig. 1, the dynamics of a typical artificial neuron used in DNNs differ greatly from their spiking neuron counterparts, therefore the well-known BP algorithm cannot be directly applied to DeepSNNs due to the non-differentiable nature of the neuron spiking activity, exploding gradients and dead neurons problems. All these issues will be discussed in more depth in the following.

Consider a fully connected DeepSNN. For simplicity, each neuron is assumed to emit at most one spike. In general, the membrane potential V_j^l of neuron j in layer l with N presynaptic connections can be expressed as,

$$V_j^l(t) = \sum_i^N \omega_{ij}^l \varepsilon(t - t_i^{l-1}) - \eta(t - t_j^l) \tag{1}$$

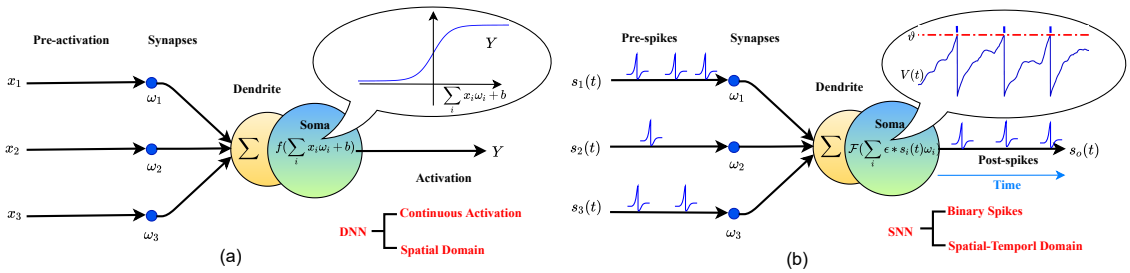


Fig. 1: Neuron models in DNNs and SNNs. (a) A typical DNN neuron model, in which the information from previous layer arrives in the form of real values in the spatial domain. x, ω, b, and Y are input activation, synaptic weights, bias and output activation, respectively. Output Y is produced by the differentiable activation function $f(\cdot)$. (b) A typical spiking neuron model, in which information from previous layer arrives in the form of spatial-temporally distributed spike events. $s_i(t)$, ω, and $s_o(t)$ are input spikes, synaptic weights and output spikes, respectively. The non-differentiable function $\mathcal{F}(\cdot)$ generates output spikes $s_o(t)$ from the membrane potential $V(t)$.

where t_i^{l-1} is the spike of the ith neuron in layer l-1, and ω_{ij}^l is the synaptic weight of the connection from neuron i (in l-1 layer) to neuron j (in l layer). Each incoming spike from neuron i will induce a postsynaptic potential (PSP) at neuron j, and the kernel $\varepsilon(t-t_i^{l-1})$ is used to describe the PSP generated by the spike t_i^{l-1}. Hence each input spike makes a contribution to the membrane potential of the neuron as described by $\omega_{ij}^l\varepsilon(t-t_i^{l-1})$ in Eq. 1. There are several PSP functions, and a commonly used one is alpha function which is defined as

$$\varepsilon(t) = \frac{t}{\tau}\exp\big(1-\frac{t}{\tau}\big) \qquad t>0 \tag{2}$$

Fig. 2(a) shows the alpha-PSP response function. As shown in Fig. 2(b), integrating the weighted PSPs gives the dynamics of the membrane potential $V_j^l(t)$. The neuron j will emit a spike when its membrane potential $V_j^l(t)$ reaches the firing threshold ϑ, as mathematically defined in the spike generation function $\mathcal{F}$:

$$t_j^l = \mathcal{F}\left\{t|V_j^l(t)=\vartheta, t\geq 0\right\} \tag{3}$$

Once a spike is emitted, the refractory kernel $\eta(t-t_j^l)$ is used to reset the membrane potential to resting.

To train SNNs using BP, we need to compute the derivative of the postsynaptic spike time t_j^l with respect to a presynaptic spike time t_i^{l-1} and synaptic weight ω_{ij}^l of the corresponding connection:

$$\frac{\partial t_j^l}{\partial \omega_{ij}^l} = \frac{\partial t_j^l}{\partial V_j^l(t_j^l)}\frac{\partial V_j^l(t_j^l)}{\partial \omega_{ij}^l} \qquad \text{if} \qquad t_j^l > t_i^{l-1} \tag{4}$$

$$\frac{\partial t_j^l}{\partial t_i^{l-1}} = \frac{\partial t_j^l}{\partial V_j^l(t_j^l)}\frac{\partial V_j^l(t_j^l)}{\partial t_i^{l-1}} \qquad \text{if} \qquad t_j^l > t_i^{l-1} \tag{5}$$

Due to the discrete nature of the spike generation function (Eq. 3), we face a challenge in solving the partial derivative $\partial t_j^l/\partial V_j^l(t_j^l)$ in Eq. 4, which we refer to as the problem of **non-differentiable spike function**. Existing spike-driven learning algorithms [43], [59] assume that the membrane potential $V_j^l(t)$ increases linearly in the infinitesimal time interval before spike time t_j. Then, $\partial t_j/V_j(t)$ can be expressed as

$$\frac{\partial t_j^l}{\partial V_j^l(t_j^l)} = \frac{-1}{\partial V_j^l(t_j^l)/\partial t_j^l} = \frac{-1}{\sum_i^N \omega_{ij}^l \frac{\partial \varepsilon(t_j^l - t_i^{l-1})}{\partial t_j^l}} \tag{6}$$

with

$$\frac{\partial \varepsilon(t_j^l - t_i^{l-1})}{\partial t_j^l} = \frac{\exp(1-(t_j^l-t_i^{l-1})/\tau)}{\tau^2}(\tau + t_i^{l-1} - t_j^l) \tag{7}$$

The **exploding gradient** problem occurs when $\partial V_j^l(t_j^l)/\partial t_j^l \approx 0$ i.e. the membrane potential is reaching the firing threshold, emitting a spike (Fig. 2b). Since $\partial V_j(t_j)/\partial t_j$ is the denominator in Eq. 6, this causes Eq. 6 to explode with large weight updates. Despite the progress made to address this problem such as adaptive learning rate [60] and dynamic firing threshold [47], the problem has not been fully resolved.

It is shown from Eq. 4 and Eq. 5 that when the presynaptic neuron does not emit a spike, the error cannot be back-propagated through $\partial V_j^l(t_j^l)/\partial t_i^{l-1}$, which then results in **dead neurons**. This problem also exists with analog neurons with ReLU activation function in DNNs. However, due to the leaky nature of the PSP kernel and the spike generation mechanism, spiking neurons encounter a more serious dead neuron problem. As shown in Fig. 2(c), there are three input spikes, and the neuron emits a spike with large synaptic weights (blue curve). With slightly reduced synaptic weights, the membrane potential stays below the threshold hence becoming a dead neuron (green curve). When the neuron does not spike, no errors can back-propagate through it. The dead neuron problem is fatal in spike-driven learning algorithms.

III. Spiking Neuron Model and Learning Algorithm

In this section, we describe how the above challenges may be overcome, thereby a DeepSNN may still be trained using BP. To this end, we introduce Rectified Linear Postsynaptic Potential function (ReL-PSP) as a new spiking neuron model, and present a spike-timing-dependent back propagation (STDBP) learning algorithm that is based on ReL-PSP.

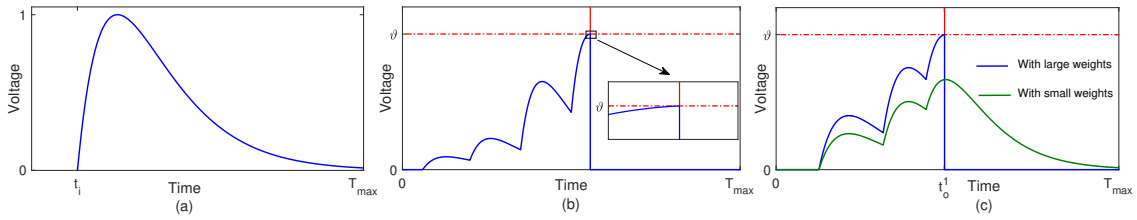


Fig. 2: (a) Alpha shape PSP function. (b) The membrane potential barely reaches the firing threshold, and exploding gradient occurs. (c) The alpha-PSP neuron with weak synaptic weights is susceptible to be a dead neuron .

A. ReL-PSP based spiking neuron model

As discussed earlier in Section II, BP cannot be directly applied in DeepSNNs due to problems of non-differentiable spike function, exploding gradient and dead neuron. To overcome these problems, we propose a simple yet efficient Rectified Linear Postsynaptic Potential (ReL-PSP) based spiking neuron model, whose dynamics is defined as follows,

$$V_j^l(t) = \sum_i^N \omega_{ij}^l K(t - t_i^{l-1}) \tag{8}$$

whereby $K(t - t_i^{l-1})$ is the kernel of the PSP function, which is defined as

$$K(t - t_i^{l-1}) = \begin{cases} t - t_i^{l-1} & \text{if} \quad t > t_i^{l-1} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

As shown in Fig. 3(a), given an input spike at t_i^{l-1}, the membrane potential after t_i^{l-1} is a linear function of time t. Since the shape of the proposed PSP function resembles that of a rectified linear function, we name it the ReL-PSP function. In the following, we will analyze how the proposed neuron model solves the above-mentioned problems.

1) Non-differentiable spike function: As shown in Fig. 3b, due to the linearity of the ReL-PSP, the membrane potential $V_j^l(t)$ increases linearly prior to spike time t_j^l. The linearity is a much desired property from a postsynaptic potential function. We can now directly use Eq. 10 to compute $\partial t_j^l / \partial V_j^l(t_j^l)$. This resolves the issue of non-differentiable spike generation.

$$\begin{aligned} \frac{\partial t_j^l}{\partial V_j^l(t_j^l)} &= -\frac{1}{\partial V_j^l(t_j^l)/\partial t_j^l} \\ &= \frac{-1}{\sum_i^N \omega_{ij}^l \frac{\partial K(t_j^l - t_i^{l-1})}{\partial t_j^l}} \\ &= \frac{-1}{\sum_i^N \omega_{ij}^l} \qquad \text{if} \quad t_j^l > t_i^{l-1} \end{aligned} \tag{10}$$

The precise gradients in BP provide the necessary information for network optimization, which is the key to the performance of DNNs. Without having to assume linearity, we use the precise value of $\partial t_j^l / \partial V_j^l(t_j^l)$ instead of approximating it, and avoid accumulating errors across multiple layers.

2) Gradient explosion: Exploding gradient occurs when the denominator in Eq. 6 approaches 0. In this case, the membrane potential just reaches the firing threshold at spike time, and is caused by the combined effect of ω_{ij}^l and partial derivative of the PSP function. As $\sum_i^N \omega_{ij}^l$ may still be close to 0, the exploding gradient problem may not be completely solved. However, from Eqs. 3 and 8, we obtain the spike time t_j^l as a function of input spikes and synaptic weights ω_{ij}^l, and the spike time t_j^l can be calculated as,

$$t_j^l = \frac{\vartheta + \sum_i^N \omega_{ij}^l t_i^{l-1}}{\sum_i^N \omega_{ij}^l} \qquad \text{if} \quad t_j^l > t_i^{l-1} \tag{11}$$

Should the $\sum_i^N \omega_{ij}^l$ be close to 0, the spike t_j^l will be emitted late, and may not contribute to the spike t_j^{l+1} in the next layer. Therefore, the neuron j in the l layer does not participate in error BP, and does not result in exploding gradient.

3) Dead neuron: In neural networks, sparse representation (few activated neurons) has many advantages, such as information disentangling, efficient variable-size representation, linear separability etc. However, sparsity may also adversely affect the predictive performance. Given the same number of neurons, sparsity reduces the effective capacity of the model [61]. Unfortunately, as shown in Fig. 2(c), due to the leaky nature of the alpha-PSP and the spike generation mechanism, such a spiking neuron is more likely to suffer from the dead neuron problem.

As shown in Fig. 3(c), with the ReL-PSP kernel, the PSP increases over time within the simulation window T_{max} until the postsynaptic neuron fires a spike . Hence the neuron with a more positive sum of weights fires earlier than one with a less positive sum, with lower probability of becoming a dead neuron. Overall, the proposed ReL-PSP greatly alleviates the dead neuron problem as the PSP does not decay over time, while maintaining a sparse representation to the same extent of the ReLU activation function.

B. Error backpropagation

Given a classification task with n categories, each neuron in the output layer is assigned to a category. When a training sample is presented to the neural network, the corresponding output neuron should fire the earliest. Several loss functions can be constructed to achieve this goal [36], [48], [49]. In this work, the cross-entropy loss function is used. To minimise the spike time of the target neuron, at the same time, maximise the spike time of non-target neurons, we use the softmax function on the negative values of the spike times in the output layer: $p_j = \exp(-t_j)/\sum_i^n \exp(-t_i)$. The loss function is given by,

$$L(g, \mathbf{t^o}) = -\ln \frac{\exp(-\mathbf{t^o}[g]))}{\sum_i^n \exp(-\mathbf{t^o}[i])} \tag{12}$$

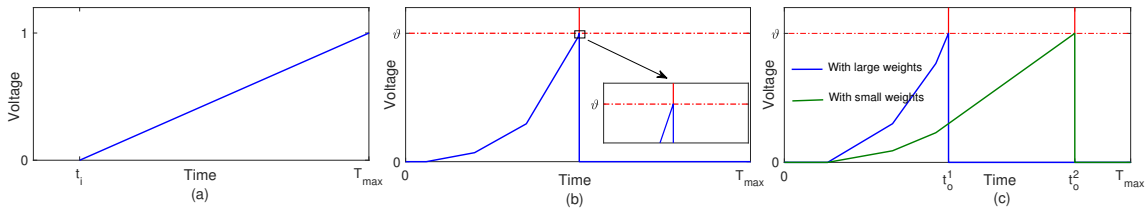


Fig. 3: There are three input spikes denoted as t_1, t_2, t_3. The blue and green lines show the membrane potential with large and small synaptic weights, respectively. (a) ReL-PSP function. (b) Trace of the neuron membrane potential during threshold crossing. (c) The ReL-PSP neuron generates spikes at t_o^1 and t_o^2 with large and small synaptic weights, respectively ($t_o^1 < t_o^2$).

where $\mathbf{t^o}$ is the vector of the spike times in the output layer and g is the target class index [36].

The loss function is minimised by updating the synaptic weights across the network. This has the effect of delaying or advancing spike times across the network. The derivatives of the first spike time t_j^l with respect to synaptic weights ω_{ij}^l and input spike times t_i^{l-1} are given by

$$\frac{\partial t_j^l}{\partial \omega_{ij}^l} = \frac{\partial t_j^l}{\partial V_j^l(t_j^l)} \frac{\partial V_j^l(t_j^l)}{\partial \omega_{ij}^l} = \frac{t_i^{l-1} - t_j^l}{\sum_i^N \omega_{ij}^l} \quad \text{if} \quad t_j^l > t_i^{l-1} \tag{13}$$

$$\frac{\partial t_j^l}{\partial t_i^{l-1}} = \frac{\partial t_j^l}{\partial V_j^l(t_j^l)} \frac{\partial V_j^l(t_j^l)}{\partial t_i^{l-1}} = \frac{\omega_{ij}^l}{\sum_i^N \omega_{ij}^l} \quad \text{if} \quad t_j^l > t_i^{l-1} \tag{14}$$

Following Eq. 13 and Eq. 14, a standard BP can be applied for DeepSNNs training.

IV. Hardware Model

To put the proposed spiking neuron model and STDBP learning algorithm into action, we evaluate their deployment on a hardware architecture for pattern classification tasks. We first implement them in spiking neural networks for inference operations on YOSO platform [58] which is specifically designed for accelerating temporal coding based SNN models with sparse spiking activity. It was shown that YOSO facilitates ultra-low-power inference operations (< 1 mW) as a neuromorphic accelerator with state-of-the-art performance.

In this section, we will describe the hardware architecture of YOSO, the technique that maps SNNs on YOSO, and the simulation methodology that evaluates the YOSO performance during inference operations.

A. Hardware Architecture

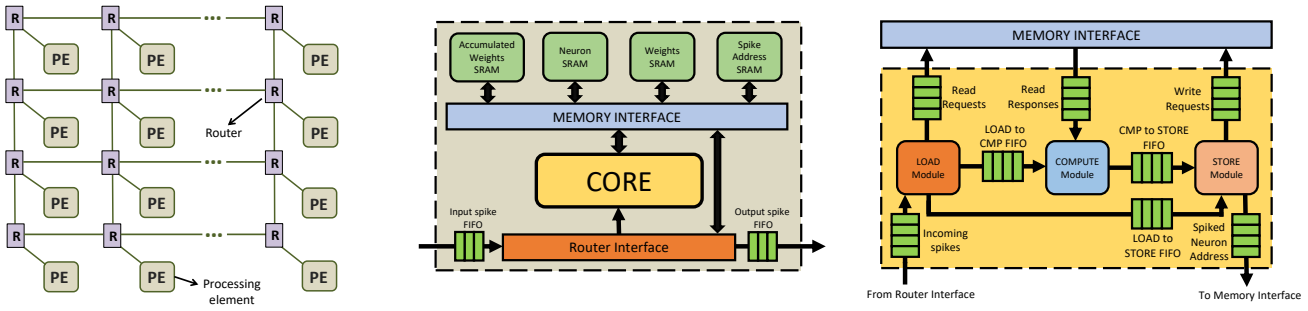
As shown in Fig. 4 (a), YOSO is a Network-On-Chip (NoC) architecture with multiple processing elements (PEs) connected in a mesh topology. Each PE has a router, which facilitates the propagation of spikes from one PE to another. As shown in Fig. 4 (b), each PE in the NoC consists of four static random-access memories (SRAMs), a memory interface, a core, a router interface, first-in-first-out (FIFO) buffers to facilitate communication between router and router interface. The router sends or receives the spikes to or from other PEs in the NoC. During the initialization phase, i.e. when the accelerator is downloading the model to be run, the router sends the spikes to SRAMs directly via router and memory interfaces. During the inference phase, the router sends the spikes to the core, where all the computations take place. Note that YOSO [58] can only accelerate inference operations and training needs to be performed offline.

The SRAMs are used to store all the information required for processing the incoming spikes and for generating the output spikes. As shown in Fig. 4 (b), the SRAMs can communicate with the core and router interface via a memory interface. The accumulated weight, neuron, weight, and spike address SRAMs store accumulated weights, neuron potentials, weights between two layers, and the spike addresses of the neurons allocated for that PE, respectively.

As shown in Fig. 4 (c), the core consists of three modules– load, compute, and store. The load module is responsible for decoding the information encoded in incoming spikes. A spike processing algorithm introduced in [58] is used to encode the information in spike packets that can read by the load module. After decoding the information in incoming spikes, the load module generates read requests to different SRAM blocks. After sending the read requests generated by one incoming spike, it transits to the idle state and waits for the next input spike. The compute module receives the data requested by the load module from the SRAMs, updates the data using the saturated adder, and sends the results to the FIFO connected to the store module. The store module receives addresses of the loaded data from the load module and the results from the compute module. Using the data and addresses received, it writes the data back to the SRAMs. It also generates spikes when a condition, depending on the chosen techniques, TTFS or softmax, is met to be sent to the next layer. In addition, the core is designed based on the decoupled access-execute model [62], which enables it to hide memory access latency by performing different computations parallelly.

B. Mapping

In this work, we plan to accelerate our trained SNN models on YOSO [58]. To map a $m \times n$ fully connected SNN layer on YOSO [58], a minimum of $C = MAX(\frac{n}{N}, \frac{m \times n}{W})$ PEs are needed where N is the maximum number of neurons that can be mapped to a single core and W is the maximum number of weights that the core can contain. The PEs are placed within a $\sqrt{C}$ by $\sqrt{C}$ grid. Each PE in the grid will be allocated to a layer or a part of layer for processing received by that layer.



(a) (b) (c)

Fig. 4: (a) The YOSO accelerator with network-on-chip (NoC) architecture, (b) the architecture of each processing element (PE) in YOSO, and (c) the computational core in each PE [58].

C. Hardware Simulation Methodology

We implemented the YOSO's NoC architecture using the OpenSMART NoC generator [63]. The X-Y routing mechanism is used to send the spike packets from one PE to another. In addition, we designed the PEs in such a way that each PE can meet the memory and computational requirements of at least 256 neurons [58]. To evaluate the performance of YOSO [58], we synthesized the hardware architecture shown in Fig. 4 using Synopsys Design Compiler version P-2019.03-SP5 targeting a 22nm technology node with a 6×7 PE configuration. Gate-level simulations are performed using the Synopsys VCS-MX K-2015.09-SP2-9 and power analysis is performed using the Synopsys PrimePower version P-2019.03-SP5.

V. Experiments

In this section, we evaluate both fully connected SNNs and convolutional SNNs on the image classification task based on the MNIST [64], Fashion-MNIST [65] and Caltech 101 face/motorbike datasets[1]. We benchmark their learning capabilities against existing spike-driven learning algorithms. In addition, we evaluate the inference speed and energy efficiency of our fully connected SNN on the YOSO neuromorphic accelerator.

A. Temporal coding

We employ an efficient temporal coding scheme that encodes information into spike timing, following the assumption that strongly activated neurons tend to fire early [66]. In practice, the input pixel intensity value is encoded into spike timing according to $t_i = \alpha(-s_i + 255)/255$, where t_i is the firing time of the ith neuron, s_i is intensity value of the ith pixel with $s_i \in [0, 255]$, and α is a scaling factor. In this way, more salient information is encoded into an earlier spike by the corresponding input neuron. These encoded spikes are propagated to subsequent layers following the dynamics of the SNN. Similar to the input layer, the neurons in the hidden and output layer that are strongly activated will fire first. As such, the temporal coding is maintained throughout the DeepSNN, and the output neuron that fires first categorizes the input sample.

[1]http: //www.vision.caltech.edu

B. MNIST Dataset

The MNIST dataset comprises of 70,000 28×28 grayscaled images, with 60,000 and 10,000 for training and testing, respectively. We first train an ANN model to initialize the the corresponding SNN. Following the temporal coding in section V-A., we encode the images into spike patterns, then use them to train a fully connected and convolutional SNN using the proposed STDBP algorithm. A complete convolutional SNN network structure is provided in Fig. 5. During training, we add jittering noise to the input spike patterns so as to improve the performance of the trained model. The jitter intervals are randomly drawn from a Gaussian distribution with zero mean and variance $\sigma_j = 0.14$. The SNN is trained for 150 epochs using the Adam optimizer, with a batch size of 128. The hyperparameters β_1 and β_2 of the Adam optimizer are set to 0.9 and 0.999, respectively. The learning rate starts at 0.0002 and gradually decreases to 0.00005 by the end of the training.

As the experimental results summarised in Table I, the proposed STDBP learning algorithm could reach accuracies of 98.1% and 98.5% with the network structures of 784-400-10 and 784-800-10, respectively. They outperform previously reported results of SNNs with the same network structures. For example, with the structure of 784-400-10, the classification accuracy of our method is 98.1%, while the accuracy achieved by Mostafa [36] is 97.5%. Another advantage of our algorithm is that it does not need additional training strategies, such as constraints on weights and gradient normalization, which are widely used in previous works to improve their performance [36], [48], [49]. This facilitates large-scale implementation of STDBP, and makes it possible to train more complex CNN structure. The proposed convolutional SNN achieves an accuracy of 99.4%, much higher than all the results obtained by the fully connected SNNs. To our best knowledge, this is the first implementation of a convolutional SNN structure with the single-spike-timing-based supervised learning algorithm.

Fig. 6 shows the distribution of spike timing in the hidden layers and of the earliest spike time in the output layer across 10,000 test images for two SNNs, namely 784-400-10 and 784-800-10. In both cases, the SNN makes a decision after only a fraction of hidden layer neurons are activated. For the 784-400-10 topology, an output neuron spikes (a class is selected) after only 48.6% of the hidden neurons have spiked.

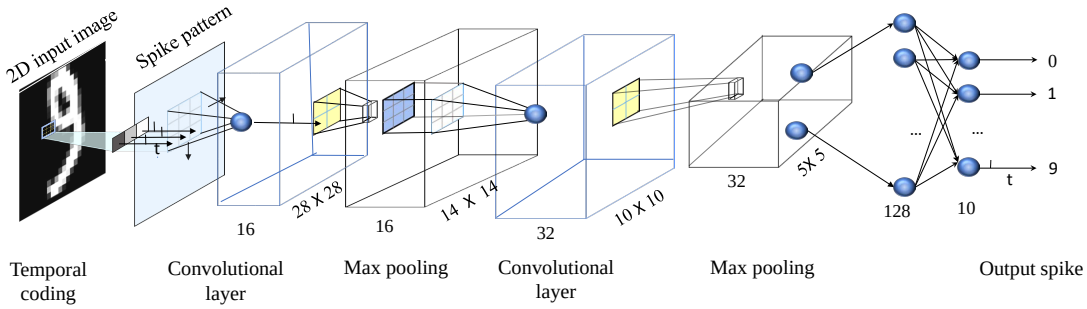


Fig. 5: The convolutional spiking neural network used in this work.

TABLE I: The classification accuracies of existing spike-driven learning algorithms on the MNIST dataset. We use the following notation to indicate the SNN architecture. Layers are separated by - and spatial dimensions are separated by $\times$. The convolution layer and pooling layer are represented by C and P, respectively.

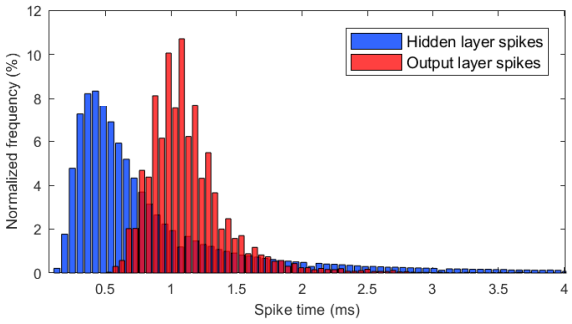
Model	Coding	Network Architecture	Additional Strategy	Acc. (%)
Mostafa [36]	Temporal	784-800-10	Weight and Gradient Constraint	97.5
Tavanaei et al [67]	Rate	784-1000-10	None	96.6
Comsa et al [49]	Temporal	784-340-10	Weight and Gradient Constraint	97.9
Kheradpisheh et al [48]	Temporal	784-400-10	Weight Constraint	97.4
ANN	Rate	784-800-10	None	98.6
STDBP (This work)	Temporal	784-340-10	None	**98.0**
STDBP (This work)	Temporal	784-400-10	None	**98.1**
STDBP (This work)	Temporal	784-800-10	None	**98.5**
STDBP (This work)	Temporal	784-1000-10	None	**98.5**
CNN	Rate	28×28-16C5-P2-32C5 -P2-800-128-10	None	99.5
STDBP (This work)	Temporal	28×28-16C5-P2-32C5 -P2-800-128-10	None	**99.4**

The network is thus able to make rapid decisions about the input class. In addition, during the simulation time, only 66.3% of the hidden neurons have spiked. Therefore, the experimental results suggest that the proposed learning algorithm works in an accurate, fast and sparse manner.

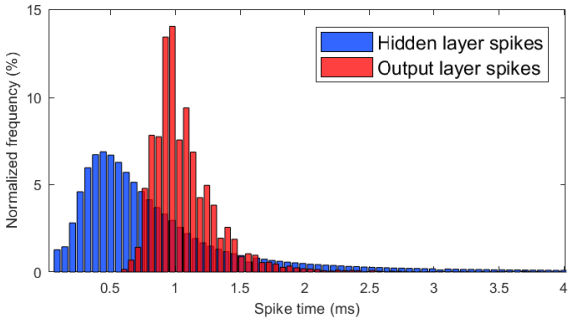
To investigate whether the proposed ReL-PSP solves the problems of exploding gradient and dead neuron, we take the fully connected SNNs as an example and 20 independent experiments are conducted with different initial synaptic weights. The data distribution of error gradients, learned synaptic weights and dead neuron counts are reported in Fig. 7, Fig. 8 and Fig. 9, respectively.

Fig. 7(a) shows the error gradients of the SNN with an alpha-PSP function, where gradients generally take values that are much larger than those in ReL-PSP (Fig. 7(b)) and ANNs (Fig. 7(c)). The results confirm our findings in the problem analysis in section II, that is, the alpha-PSP function is prone to the gradient exploding problem. We note that the previous studies only partly alleviate this problem, for example, the adaptive learning rate [60] and dynamic firing threshold [47] methods. We are encouraged to see that the gradients of the ReL-PSP function follows a normal distribution that is close to that of ANN, thus ensuring a stable gradient propagation in DeepSNNs. In addition, as shown in Fig. 8(a), the gradient exploding problem also leads to skewed weights, which may adversely affect the performance of the trained model, and is now addressed by the ReL-PSP function.

For artificial neural networks, typically only a subset of neurons are activated at the same time [61]. We note that excessive inactivation will lead to the dead neuron problem, which reduces the effective capacity of the network and thus the predictive capability. To assess the severity of the dead neuron problem, we record the percentage of dead neurons during the training process, and report both the mean and the standard deviations across 20 independent runs in Fig. 9. As the training progresses, the percentage of dead neurons increases across both SNNs and ANN. After 100 training epochs, the SNN with ReL-PSP has 30% active neurons and achieves a test accuracy of 98.5%. In contrast, the SNN with alpha-PSP only has 5% active neurons with a test accuracy of 92.5%. As we discussed in section III-A, due to the leaky nature of traditional alpha-PSP function and the spike generation mechanism, SNNs with alpha-PSP function are more likely to suffer from the dead neuron problem. The experimental results corroborate our hypothesis and suggest that ReL-PSP function effectively addresses this problem. After 100 training epochs, the ANN has about 40% active neurons with an accuracy of 98.6%. Overall, the results show that the SNN model with ReL-PSP achieves competitive results with sparse neuronal activities.



(a)



(b)

Fig. 6: Histograms of spike times in the hidden layers and the output layer across 10,000 test images for the two SNNs: (a) 784-400-10 and (b) 784-800-10.

C. Fashion-MNIST dataset

Fashion-MNIST dataset [65] has same number of training/testing samples as MNIST, but is more challenging than MNIST. The samples are associated with 10 different classes like T-shirt/top, Pullover, Trouser, Dress, Sandal, Coat, Shirt, Bag, Sneaker and Ankle boot. Here we use Fashion-MNIST to compare our method with the existing fully-connected feedforward SNNs and convolutional SNNs.

Table II shows the classification accuracies and characteristics of different methods on Fashion-MNIST dataset. The proposed learning algorithm still deliver the best test accuracy in temporal coding based SNN methods. For example, our method can achieve accuracies of 88.1% and 90.1% with the fully-connected SNN and convolutional SNN, respectively. These results outperforms the best reported result of 88.0% [48] in temporal coding based SNN models.

D. Caltech face/motorbike dataset

In this experiment, the performance of the proposed STDBP is evaluated on the face/motobike categories of the Caltech 101 dataset (http://www.vision.caltech.edu). For each category, the training and validation set consist of 200 and 50 randomly selected samples, respectively, and the others are regard as testing samples. Before training, all images are rescaled and converted to 160×250 grayscale images. Fig. 10 shows some samples of the converted images. The converted images are then encoded into spike patterns by temporal coding.

The classification accuracies of different SNN-based computational models are shown in Table III. The proposed STDBP learning algorithm achieves an accuracy of 99.2% with the fully connected SNN structure and an accuracy of 99.5% with the convolutional SNN structure. The accuracy obtained by STDBP outperforms the previously reported SNN-based methods on this dataset. For example, In Kheradpisheh et al [74], an convolutional SNN structure with a SVM classifier achieves an accuracy of 99.1% on the same dataset. Moreover, it is not a fully spike-based computational model that the membrane potential is used as the classification signal. Recently, a spike-based fully connected SNNs model achieves an accuracy of 99.2%. However, the proposed method with convolutional SNN structure reaches an accuracy of 99.5%, which is the state-of-the-art performance on this benchmark.

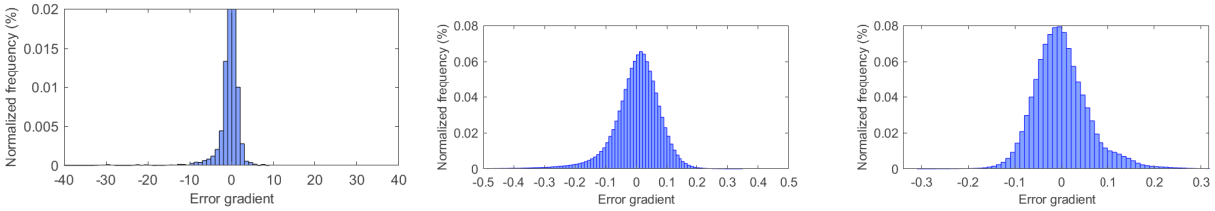
E. Hardware Simulation Results

We will now focus on evaluating the power and energy efficiency of our SNN models by accelerating their inference operations on YOSO. As a baseline for comparison with other neuromorphic accelerators, we considered a fully connected SNN with network architecture 784-800-10 and trained it on MNIST data with proposed STDBP learning algorithm. The learned weights are then transferred to YOSO for accelerating the inference operations. As shown in Table IV, YOSO consumes 0.751 mW of total power consumption and 47.71 ms of latency to classify an image from MNIST dataset. In addition, YOSO achieves 399×, 159.8×, 143.8×, and 1.67× power savings as compared to Spinnaker [56], Tianji [57], TrueNorth-b [24], and Shenjing [55], respectively. Though, TrueNorth-a consumes less power than YOSO, there is a significant difference between the classification accuracies of TrueNorth-a and YOSO (See Table IV). Moreover, YOSO provides 108.7×, 6×, and 3× energy efficiency as compared to Spinnaker [56], SNNwt [76], and TrueNorth-b [24], respectively.

VI. Discussion and Conclusion

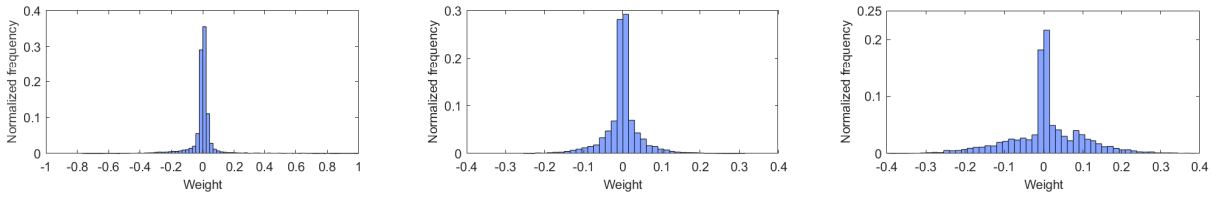
In this work, we analysed the problems that BP faces in a DeepSNN, namely the non-differentiable spike function, the exploding gradient, and the dead neuron problems. To address these issues, we proposed the Rectified Linear Postsynaptic Potential function (ReL-PSP) for spiking neurons and the STDBP learning algorithm for DeepSNNs. We evaluated the proposed method on both a multi-layer fully connected SNN and a convolutional SNN. The conducted experiments on MNIST showed an accuracy of 98.5% in the case of the the fully connected SNN and 99.4% with the convolutional SNN, which is the state-of-art in spike-driven learning algorithms for DeepSNNs.

There have been a number of learning algorithms for DeepSNNs, such as conversion methods [24]–[33], and surrogate gradients methods [15], [37]–[41]. These methods are not compatible with temporal coding and spike-based learning mechanism. To overcome the non-differentiability of spike function, many methods have been studied [36], [43]–[49] to facilitate backpropagation. Two common drawbacks of these methods are exploding gradients and dead neurons,



(a) SNN with alpha-PSP function (b) SNN with ReL-PSP function (c) ANN

Fig. 7: Distribution of error gradients of different methods.



(a) SNN with alpha-PSP function (b) SNN with ReL-PSP function (c) Learned weights of ANN

Fig. 8: Distribution of learned weights of different methods.

TABLE II: The classification accuracy (Acc.) of the existing SNN-based computational models on the Fashion-MNIST dataset.

Model	Coding	Network Architecture	Acc. (%)
S4NN [48]	Temporal	784-1000-10	88.0
BS4NN [68]	Temporal	784-1000-10	87.3
Hao et al. [69]	Rate	784-6000-10	85.3
Zhang et al. [70]	Rate	784-400-400-10	89.5
STDBP (This work)	Temporal	784-1000-10	88.1
Ranjan et al. [71]	Rate	28×28-32C3-32C3-P2-128-10	89.0
STDBP (This work)	Temporal	28×28-16C5-P2-32C5-P2-800-128-10	90.1

TABLE III: The classification accuracy (Acc.) of existing SNN-based computational models on the Caltech face/motorbike dataset.

Model	Learning method	Network Architecture	Classifier	Acc. (%)
Masquelier et al. [72]	Unsupervised STDP	HMAX [73]	RBF	99.2
Kheradpisheh et al. [74]	Unsupervised STDP	28×28-4C5-P7-20C16-P2-10C5	SVM	99.1
Mozafari et al. [75]	Reward modulated STDP	HMAX [73]	Spike-based	98.2
Kheradpisheh et al. [48]	Spike-based backpropagation	$160\times250-4-2$	Spike-based	99.2
STDBP (This work)	Spike-based backpropagation	$160\times250-4-2$	Spike-based	**99.2**
STDBP (This work)	Spike-based backpropagation	28×28-16C5-P2-32C5-P2-800-128-10	Spike-based	**99.5**

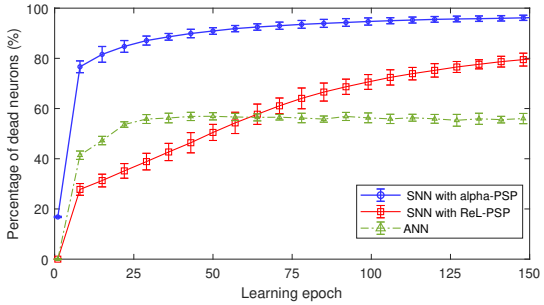


Fig. 9: The percentage of dead neurons during the training process. The error bars denote the standard deviations across 20 independent experiments.

Fig. 10: Some samples of the converted images from Caltech 101 face/motorbike dataset

which have been partially addressed using techniques such as constraints on weights and gradient normalization. These techniques affect learning efficiency, thus limiting the scope of

TABLE IV: Comparison of a fully connected SNN with various neuromorphic accelerators on the MNIST dataset sorted by the accuracy. Acc. denotes Top-1 accuracy, frame rate is reported as frames per second (fps), Tech is CMOS technology node in nm, power in mW.

Accelerator	Coding	Acc. (%)	fps	Tech	Power	uJ/frame
SNNwt [76]	Rate	91.82	-	65	-	214.700
TrueNorth-a [24]	Rate	92.70	1000	28	0.268	0.268
TrueNorth-b [24]	Rate	99.42	1000	28	108.000	108.000
Loihi-a [77]	Rate	98.79	120	14	-	-
Loihi-b [78]	Rate	99.21	150	14	99.248	660
Spinnaker [56]	Rate	95.01	77	130	300.000	3896.000
Tianji [57]	Rate	96.59	-	120	120.000	-
Shenjing [55]	Rate	96.11	40	28	1.260	38.000
STDBP+YOSO (This work)	Temp.	98.45	21	22	**(0.878*) 0.751**	**(41.93*) 35.839**

**Scaled for 28 nm process ($\times$1.17 for half a generation)

their applications in large-scale networks. The proposed STDBP learning algorithm with ReL-PSP spiking neuron model can train DeepSNNs directly without any additional technique, hence allowing the DeepSNNs to scale, as shown in the high accuracy of the convolutional SNN.

Other than what is discussed in this paper, the proposed ReL-PSP neuron model and STDBP have other attributes that might make it more energy-efficient and (neuromorphic) hardware friendly. Firstly, the linear ReL-PSP function is simpler than alpha-PSP for hardware implementation. Secondly, unlike rate-based encoding methods that require more time to generate enough output spikes for classification, our method takes advantage of temporal coding and uses a single spike, which is more sparse and energy-efficient, given energy is mainly consumed during spike generation and transmission. Thirdly, without additional training techniques, on-chip training in neuromorphic chips would be much easier to realize. Finally, even if the training is performed offline, inference of our SNN models can be accelerated with much less power and energy consumption as compared to other state-of-the-art neuromorphic accelerators.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *Proc. Interspeech 2019*, pp. 3465–3469, 2019.

[4] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[6] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[7] J. Feldmann, N. Youngblood, C. Wright, H. Bhaskaran, and W. Pernice, "All-optical spiking neurosynaptic networks with self-learning capabilities," *Nature*, vol. 569, no. 7755, p. 208, 2019.

[8] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421.

[9] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[10] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[11] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in neuroscience*, vol. 12, 2018.

[12] X. Li, H. Liu, F. Xue, H. Zhou, and Y. Song, "Liquid computing of spiking neural network with multi-clustered and active-neuron-dominant structure," *Neurocomputing*, vol. 243, pp. 155–165, 2017.

[13] A. Zhang, H. Zhou, X. Li, and W. Zhu, "Fast and robust learning in spiking feed-forward neural networks based on intrinsic plasticity mechanism," *Neurocomputing*, vol. 365, pp. 102–112, 2019.

[14] V. P. K. Miriyala and M. Ishii, "Ultra-low power on-chip learning of speech commands with phase-change memories," *arXiv preprint arXiv:2010.11741*, 2020.

[15] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *arXiv preprint arXiv:1901.09948*, 2019.

[16] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, "Rethinking the performance comparison between snns and anns," *Neural Networks*, vol. 121, pp. 294–307, 2020.

[17] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, 2018.

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

[19] J. Wu, Y. Chua, M. Zhang, Q. Yang, G. Li, and H. Li, "Deep spiking neural network with spike count based learning rule," *arXiv preprint arXiv:1902.05705*, 2019.

[20] J. Wu, C. Xu, D. Zhou, H. Li, and K. C. Tan, "Progressive tandem learning for pattern recognition with deep spiking neural networks," *arXiv preprint arXiv:2007.01204*, 2020.

[21] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.

[22] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, "Comprehensive snn compression using admm optimization and activity regularization," *arXiv preprint arXiv:1911.00822*, 2019.

[23] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes," *Nature Machine Intelligence*, pp. 1–9, 2021.

[24] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[25] E. Hunsberger and C. Eliasmith, "Spiking deep networks with lif neurons," *arXiv preprint arXiv:1510.08829*, 2015.

[26] S. K. Essera, P. A. Merollaa, J. V. Arthura, A. S. Cassidya, R. Appuswamya, A. Andreopoulosa, D. J. Berga, J. L. McKinstrya, T. Melanoa, D. R. Barcha *et al.*, "Convolutional networks for fast energy-efficient neuromorphic computing," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 41, pp. 11 441–11 446, 2016.

[27] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers in neuroscience*, vol. 7, p. 178, 2013.

[28] Q. Liu, Y. Chen, and S. Furber, "Noisy softplus: an activation function that enables snns to be trained as anns," *arXiv preprint arXiv:1706.03609*, 2017.

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

[30] P. U. Diehl, B. U. Pedroni, A. Cassidy, P. Merolla, E. Neftci, and G. Zarrella, "Truehappiness: Neuromorphic emotion recognition on truenorth," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 4278–4285.

[31] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[32] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[33] B. Han, G. Srinivasan, and K. Roy, "Rmp-snns: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural networks," *arXiv preprint arXiv:2003.01811*, 2020.

[34] S. Deng and S. Gu, "Optimal conversion of conventional artificial neural networks to spiking neural networks," *arXiv preprint arXiv:2103.00476*, 2021.

[35] J. Wu, Y. Chua, M. Zhang, G. Li, H. Li, and K. C. Tan, "A hybrid learning rule for efficient and rapid inference with spiking neural networks," *arXiv preprint arXiv:1907.01167*, 2019.

[36] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017.

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

[38] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[39] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018.

[40] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, 2018.

[41] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, and L. Shi, "Direct training for spiking neural networks: Faster, larger, better," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1311–1318.

[42] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *NeurIPS*, 2018.

[43] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[44] S. B. Shrestha and Q. Song, "Robust spike-train learning in spike-event based weight update," *Neural Networks*, vol. 96, pp. 33–46, 2017.

[45] Y. Xu, X. Zeng, L. Han, and J. Yang, "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks," *Neural Networks*, vol. 43, pp. 99–113, 2013.

[46] S. B. Shrestha and Q. Song, "Robustness to training disturbances in spikeprop learning," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3126–3139, 2017.

[47] C. Hong, X. Wei, J. Wang, B. Deng, H. Yu, and Y. Che, "Training spiking neural networks for cognitive tasks: A versatile framework compatible with various temporal codes," *IEEE transactions on neural networks and learning systems*, 2019.

[48] S. R. Kheradpisheh and T. Masquelier, "S4nn: temporal backpropagation for spiking neural networks with one spike per neuron," *arXiv preprint arXiv:1910.09495*, 2019.

[49] I. M. Comsa, K. Potempa, L. Versari, T. Fischbacher, A. Gesmundo, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function," *arXiv preprint arXiv:1907.13223*, 2019.

[50] C.-K. Lin, A. Wild, G. Chinya, M. Davies, N. Srinivasa, D. Lavery, and H. Wang, "Programming spiking neural networks on Intel Loihi," *Computer*, vol. 51, no. 3, pp. 52–61, Mar. 2018.

[51] J. Pei, L. Deng, S. Song, M. Zhao, Y. Zhang, S. Wu, G. Wang, Z. Zou, Z. Wu, W. He *et al.*, "Towards artificial general intelligence with hybrid tianjic chip architecture," *Nature*, vol. 572, no. 7767, p. 106, 2019.

[52] L. Deng, G. Wang, G. Li, S. Li, L. Liang, M. Zhu, Y. Wu, Z. Yang, Z. Zou, J. Pei *et al.*, "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 8, pp. 2228–2246, 2020.

[53] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," vol. 34, no. 10, Oct 2015, pp. 1537–1557.

[54] D. Neil and S. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[55] B. Wang, J. Zhou, W.-F. Wong, and L.-S. Peh, "Shenjing: A low power reconfigurable neuromorphic accelerator with partial-sum and spike networks-on-chip," *2020 Proceedings of Design, Automation, and Test in Europe (DATE), Grenoble, France*, Mar 2020.

[56] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IJCNN)*, Jun 2008, pp. 2849–2856.

[57] Y. Ji, Y. Zhang, W. Chen, and Y. Xie, "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar 2018, pp. 448–460.

[58] P. Srivatsa, T. N. C. Kyle, T. Yaswanth, W. Jibin, Z. Malu, L. Haizhou, and T. E. Carlson, "You only spike once: Improving energy-efficient neuromorphic inference to ann-level accuracy," pp. 1–8, 2020. [Online]. Available: http://workshops.inf.ed.ac.uk/accml/papers/2020-isca/2nd_AccML_paper_1.pdf

[59] Q. Yu, H. Li, and K. C. Tan, "Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity," *IEEE transactions on cybernetics*, vol. 49, no. 6, pp. 2178–2189, 2018.

[60] S. B. Shrestha and Q. Song, "Adaptive learning rate of spikeprop based on weight convergence analysis," *Neural Networks*, vol. 63, pp. 185–198, 2015.

[61] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[62] J. E. Smith, "Decoupled access/execute computer architectures," in *Proceedings of the 9th Annual Symposium on Computer Architecture (ISCA)*, Apr. 1982, p. 112–119.

[63] H. Kwon and T. Krishna, "OpenSMART: Single-cycle multi-hop noc generator in BSV and Chisel," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2017, pp. 195–204.

[64] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[65] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[66] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural networks*, vol. 14, no. 6-7, pp. 715–725, 2001.

[67] A. Tavanaei and A. Maida, "Bp-stdp: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[68] S. R. Kheradpisheh, M. Mirsadeghi, and T. Masquelier, "Bs4nn: Binarized spiking neural networks with temporal coding and learning," *arXiv preprint arXiv:2007.04039*, 2020.

[69] Y. Hao, X. Huang, M. Dong, and B. Xu, "A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule," *Neural Networks*, vol. 121, pp. 387–395, 2020.

[70] W. Zhang and P. Li, "Temporal spike sequence learning via backpropagation for deep spiking neural networks," *arXiv preprint arXiv:2002.10085*, 2020.

[71] J. A. K. Ranjan, T. Sigamani, and J. Barnabas, "A novel and efficient classifier using spiking neural network," *The Journal of Supercomputing*, pp. 1–16, 2019.

[72] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, 2007.

[73] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature neuroscience*, vol. 2, no. 11, pp. 1019–1025, 1999.

[74] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[75] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 6178–6190, 2018.

[76] Z. Du, D. D. B.-D. Rubin, Y. Chen, L. Hel, T. Chen, L. Zhang, C. Wu, and O. Temam, "Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2015, pp. 494–507.

[77] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Sep 2020, pp. 1–9.

[78] B. Rueckauer, C. Bybee, R. Goettsche, M. j. Singh, Yashwardhan, and A. Wild, "Nxtf: An api and compiler for deep spiking neural networks on intel loihi," *arXiv preprint arXiv:2101.04261v1*, 2021.