Complex & Intelligent Systems
https://doi.org/10.1007/s40747-023-00983-y

ORIGINAL ARTICLE

BPLC + NOSO: backpropagation of errors based on latency code with neurons that only spike once at most

Seong Min Jin[1] · Dohun Kim[2] · Dong Hyung Yoo[1] · Jason Eshraghian[3] · Doo Seok Jeong[1]

Received: 21 October 2022 / Accepted: 22 January 2023

Abstract

For mathematical completeness, we propose an error-backpropagation algorithm based on latency code (BPLC) with spiking neurons conforming to the spike–response model but allowed to spike once at most (NOSOs). BPLC is based on gradients derived without approximation unlike previous temporal code-based error-backpropagation algorithms. The latency code uses the spiking latency (period from the first input spike to spiking) as a measure of neuronal activity. To support the latency code, we introduce a minimum-latency pooling layer that passes the spike of the minimum latency only for a given patch. We also introduce a symmetric dual threshold for spiking (i) to avoid the dead neuron issue and (ii) to confine a potential distribution to the range between the symmetric thresholds. Given that the number of spikes (rather than timesteps) is the major cause of inference delay for digital neuromorphic hardware, NOSONets trained using BPLC likely reduce inference delay significantly. To identify the feasibility of BPLC + NOSO, we trained CNN-based NOSONets on Fashion-MNIST and CIFAR-10. The classification accuracy on CIFAR-10 exceeds the state-of-the-art result from an SNN of the same depth and width by approximately 2%. Additionally, the number of spikes for inference is significantly reduced (by approximately one order of magnitude), highlighting a significant reduction in inference delay.

Keywords Backpropagation based on latency code · Spiking neural networks · Minimum-latency pooling · Symmetric dual threshold

Introduction

Spiking neural networks (SNNs) of layer-wise feedforward structure can process and convey data forward based on asynchronous spiking events without forward locking unlike feedforward deep neural networks (DNNs) [10,32]. When implemented in asynchronous neuromorphic hardware, SNNs are believed to leverage their processing efficiency. Nevertheless, asynchronous neuromorphic hardware often suffers from traffic congestion due to a large number of spikes (events) that are routed to their destination neurons through network-on-chip with limited bandwidth [9]. In this regard, the number of synaptic operations per second (SynOPS) is considered as a crucial measure of neuromorphic hardware performance, and attempts have been made to improve this synaptic operation speed to further accelerate the inference process [8,12,27,28]. Algorithm-wise approaches to improve the inference speed include the development of learning algorithms that support the inference process using fewer spikes.

Seong Min Jin and Dohun Kim have contributed equally to this work.

✉ Doo Seok Jeong
dooseokj@hanyang.ac.kr

Seong Min Jin
jin.seongmin0709@gmail.com

Dohun Kim
star007kdh@gmail.com

Dong Hyung Yoo
yoees@hanyang.ac.kr

Jason Eshraghian
jeshragh@ucsc.edu

1 Hanyang University, 222 Wangsimni-ro, Seongdong-gu, Seoul 04763, Republic of Korea

2 Samsung Advanced Institute of Technology, 130 Samsung-ro, Yeongtong-gu, Suwon-si 16678, Republic of Korea

3 University of California, Santa Cruz, Engineering Loop, Santa Cruz 95064, CA, USA

Given the limited accessibility to global data in multi-core neuromorphic hardware, learning algorithms of locality are favored as on-chip learning algorithms. However, learning algorithms of locality, e.g., naive Hebb rule [15], spike timing-dependent plasticity [4], and Ca-signaling model [21], fail to achieve high performance. Currently, it appears that the trend is moving toward off-chip learning, allowing the learner to access large global data within the general framework of error-backpropagation (backprop). The advantage is such that enriched optimization techniques for DNNs can readily be applied to SNNs, which significantly improves the performance of SNNs [10]. Nevertheless, the notable inconsistency between DNNs and SNNs lies in the fact that output spikes are non-differentiable unlike activation functions.

As a workaround, the gradients of spikes are often approximated to heuristic functions, which are popularly referred to as surrogate gradients [2,11,34,38,44]. Using surrogate gradients, the gradient values are available disregarding the presence of events, avoiding the dead neuron issue that hinders the network from learning. To date, various surrogate gradients have been proposed, e.g., boxcar function [38], arctan function [11], exponential function [34]; these methods remove the inconsistency between DNNs and SNNs, yielding the state-of-the-art classification accuracy on various datasets. Despite the technical success, such heuristic surrogate gradient methods lack theoretical completeness given the lack of theoretical foundations of surrogate gradients.

Spike timing-based backprop (temporal backprop) algorithms can avoid such surrogate gradients because the spike timing may be differentiable with the membrane potential using a linear approximation of near-threshold potential evolution [5]. Temporal backprop is generally prone to learning failure because of limited error-backpropagation paths. This is because spike timing gradients are available only for the neurons that spike at a given timestep unlike surrogate gradients. The number of error-backpropagation paths is further limited by dead neurons, i.e., neurons whose current fan-in weights are low so that they no longer fire spikes. STDBP, a temporal backprop algorithm, uses a rectified linear potential kernel to avoid the dead neuron issue [46]. The rectified linear kernel causes a monotonous increase in potential upon receiving an input spike with a positive weight, suggesting that the neurons eventually fire spikes. TSSL-BP considers additional error-backprogation paths via spikes from the same neuron to avoid learning failure due to limited error-backpropagation paths [48]. The timing gradient is calculated using the linear approximation by Bohte et al. [5]. Another temporal backprop algorithm (STiDi-BP) uses a piece-wise linear kernel to approximate the spike timing gradient to a simple function, and thus to reduce the computational cost [25,26].

Because spike timing gradients are available only for the neurons that spike, generally, the larger the number of spikes, the richer the error-backpropagation paths. Thus, more spikes are desired for a better training. However, this causes a considerable inference delay when implemented in digital neuromorphic hardware because of its limited synaptic operation speed. Concerning the desires for

- theoretically seamless applications of temporal backprop to SNNs,
- workaround for the dead neuron issue,
- fewer spikes for fast inference,

we propose a novel learning algorithm based on the spiking latency code of neurons that only spike once at most (NOSOs). NOSOs are based on the spike–response model (SRM) [13] but with an infinite hard refractory period to avoid additional spikes. The algorithm is based on the backpropagation of errors evaluated using the spiking latency code (BPLC). The key to BPLC + NOSO is such that, when spiking, spiking latency (rather than spike itself) is the measure of the response to a given input, which is differentiable without approximations unlike [5]. Thus, BPLC + NOSO is mathematically rigorous such that all required gradients are derived analytically. Other important features of BPLC + NOSO are as follows.

- The use of NOSOs for both learning and inference minimizes the workload on the event-routing circuits in neuromorphic hardware.
- To support the latency code, NOSONet includes minimum-latency pooling (MinPool) layers (instead of MaxPool or AvgPool) that pass the event of the minimum latency only for a given patch.
- Each NOSO is given two symmetric thresholds ($-\vartheta$ and ϑ) for spiking to confine the potential distribution to the range between the symmetric thresholds.
- BPLC + NOSO fully supports both folded and unfolded NOSONets, allowing us to use the automatic differentiation framework [31].

The primary contributions of this study include the following:

- We introduce a novel learning algorithm based on the spiking latency code (BPLC + NOSO) with full derivations of the primary gradients without approximations.
- We provide novel and essential methods for BPLC + NOSO support, such as MinPool layers and symmetric dual threshold for spiking, which greatly improve accuracy and inference efficiency.
- We introduce a method to quickly calculate wallclock time for inference on general digital neuromorphic hardware,

which allows a quick estimation of the inference delay for a given fully trained SNN.

The rest of the paper is organized as follows— Section "Related work" briefly overviews previous learning algorithms based on temporal codes. Section "Preliminaries" addresses primary techniques employed in BPLC + NOSO. Section "BPLC with spike response model" is dedicated to the theoretical foundations of BPLC + NOSO. Section "Experiments" addresses the performance evaluation of BPLC + NOSO on Fashion-MNIST and CIFAR-10 and effects of MinPool and symmetric dual threshold for spiking on learning efficacy. Section "Discussion" discusses the estimation of inference time for an SNN mapped onto a general digital multicore neuromorphic processor. Finally, Section "Conclusion and outlook" concludes our study.

Related work

Spike timing gradient approximation: Temporal backprop algorithms frequently use linear approximated spike timing gradients proposed by Bohte et al. [5]. The specific form of the gradient depends on the membrane potential kernel used. Bohte et al. [5], Comsa et al. [7], and Kim et al. [19] used an alpha kernel as an approximation of the genuine SRM kernel, and the corresponding gradients were evaluated using the linear approximation. Zhang et al., employed a rectified linear kernel to avoid the dead neuron issue [46] while Mirsadeghi et al., employed a piece-wise linear kernel for simple calculations of the gradient [25,26]. To apply the linear approximation by Bohte et al. [5], the gradient of membrane potential at the spike timing should be available. Integrate-and-fire (IF) neurons do not allow the gradient value at the spike timing so that Kheradpisheh and Masquelier [17] approximated the gradient to be constant at –1. The same holds for leaky integrate-and-fire (LIF) neurons. Zhang and Li [48] stated that the linear approximated was employed, but the gradient is not clearly derived.

Label-encoding as spike timings: For SNN with temporal code, the correct labels are frequently encoded as particular output spike timings [17,25,26] or the temporal order of output spikes such as time-to-first-spike (TTFS) code [30,45,46]. In the TTFS code, the neuron index of the first output spike indicates the output label.

Workaround for dead neuron: Comsa et al. proposed temporal backprop with a means to avoid dead neurons (assigning penalties to the presynaptic weights of each dead neuron) [7]. Zhang et al. [46] proposed a rectified linear potential kernel that causes a monotonous increase in potential upon receiving a spike with positive weight. Thus, the neuron eventually fire a spike. Zhang and Li [48] proposed TSSL-BP with additional backprop paths via the spikes emitted from the same neuron (intra-neuron dependency). The additional paths avoid the learning failure due to limited backprop paths by dead neurons. Kim et al. [19] combined temporal backprop paths with rate-based backprop paths to compensate for the loss of temporal backprop paths due to dead neurons.

BPLC + NOSO is clearly distinguished from the previous temporal backprop algorithms in terms of the primary perspectives addressed in this section. First, BPLC + NOSO employs no approximation for gradient evaluation unlike the previous temporal backprop algorithms including those reviewed in this section. Therefore, it barely embodies ambiguity. Second, the proposed spiking latency code is a novel data encoding scheme, distinguishable from the previous temporal code schemes. Third, the symmetric dual threshold for spiking is a novel method to avoid the dead neuron issue, which is computationally efficient since it hardly involves high-cost computations. Additionally, BPLC + NOSO is fully compatible with the original SRM without approximations.

Preliminaries

Latency code

Spiking latency is a period from the first input spike timing t_{in} and consequent spike timing $\hat{t}$ as illustrated in Fig. 1**a**. In the latency code, NOSONet encodes input data $\boldsymbol{x}$ as the spiking latency $T_{\mathrm{lat}}^{(L)}$ of the output neurons in the output layer L.

$$\boldsymbol{T}_{\mathrm{lat}}^{(L)} = \hat{\boldsymbol{t}}^{(L)} - \boldsymbol{t}_{\mathrm{in}}^{(L)} = f^{(L)}(\hat{\boldsymbol{t}}^{(L-1)}; \boldsymbol{w}^{(L-1)}), \quad (1)$$

where $\hat{\boldsymbol{t}}^{(\cdot)}$ and $\boldsymbol{t}_{\mathrm{in}}^{(\cdot)}$ denote the spike timings of the neurons in the $(\cdot)$th layer and their first input spike timings, respectively. The function $f^{(L)}$ encodes input spikes (from the layer L-1) at $\hat{\boldsymbol{t}}^{(L-1)}$ as spiking latency values $\boldsymbol{T}_{\mathrm{lat}}^{(L)}$. The larger the weight $\boldsymbol{w}^{(L-1)}$, the shorter the spiking latency $\boldsymbol{T}_{\mathrm{lat}}^{(L)}$. This latency code should be distinguished from the TTFS code [30,45,46] in which the first input spike timings $\boldsymbol{t}_{\mathrm{in}}^{(L)}$ in Eq. (1) are ignored, so that it considers the output spike timings only.

Minimum-latency pooling

The MinPool layers support the latency code. Consider the time elapsed since the first input spike, $t_{\mathrm{elap}} = t - t_{\mathrm{in}}$, for a given neuron. We consider a spiking latency map in a given 2D patch $\mathcal{D}_{\mathrm{pool}}$ at timestep t and feature (spike) map in the same patch, $\boldsymbol{T}_{\mathrm{lat},\mathcal{D}_{\mathrm{pool}}}[t]$ and $\boldsymbol{s}_{\mathcal{D}_{\mathrm{pool}}}[t]$, respectively. The latency map $\boldsymbol{T}_{\mathrm{lat},\mathcal{D}_{\mathrm{pool}}}$ is initialized to infinite values. Each

Table 1 Acronyms and symbols

Acronyms or Symbols	Description
IF	Integrate-and-fire
LIF	Leaky integrate-and-fire
SRM	Spike response model
BPLC	Error-backpropagation algorithm based on latency code
NOSO	Neurons that only spike once at most
MinPool	Minimum-latency pooling
SynOPS	Synaptic operations per second
TTFS	Time to the first spike
$\boldsymbol{T}_{\text{lat}}^{(L)}$	Spiking latency of output neurons in the output layer L
$\hat{t}_i^{(l)}$	Spike timing of the ith neuron in the lth layer
$t_{\text{in},i}^{(l)}$	First input spike timing for the ith neuron in the lth layer
$w_{ij}^{(l)}$	Synaptic weight from the jth neuron in the (l-1)th layer
$u_i^{(l)}$	Membrane potential of the ith neuron in the lth layer
$v_j^{(l)}$	Membrane potential before weight multiplication of jth neuron in the lth layer
N_n	Number of neurons in a network
N_c	Number of cores of neuromorphic processor
T_up	Time for process of multiplying the current potential by decay factor
T_sop	Time for synaptic operations at each timestep
T_inf	Inference delay

element in the map is replaced by real spiking latency when the neuron spikes. Note that the elements once replaced by real latency values are no longer overwritten because of the use of NOSOs. At time step t, MinPool outputs one if the neuron of the smallest spiking latency in the patch fires a spike, and zero otherwise.

$$x_{\min} = \arg\min_{x \in \mathcal{D}_{\text{pool}}} \left\{ \boldsymbol{T}_{\text{lat},\mathcal{D}_{\text{pool}}}[t] \right\},$$
$$\text{MinPool}\left(\mathcal{D}_{\text{pool}}\right)[t] = s_{x_{\min}}[t], \tag{2}$$

where $s_{x_{\min}}[t]$ indicates the spike function value for $x_{\min}$ at timestep t. An example of $\text{MinPool}\left(\mathcal{D}_{\text{pool}}\right)[t]\,(=1)$ is illustrated in Fig. 1**b**.

NOSO with dual threshold for spiking

Each NOSO is endowed with a symmetric dual threshold for spiking ($-\vartheta$ and ϑ), and thus a spike is generated if the membrane potential u satisfies $u \geq \vartheta$ or $u \leq -\vartheta$. Therefore, the subthreshold potential u is confined to the range between $-\vartheta$ and ϑ. The restriction on the potential offers the upper limit of potential variance over the samples in a given batch, preventing large potential variance over the samples. The symmetry in the two bounds may offer zero-mean potential over the samples. Additionally, the restriction on the potential is expected to avoid dead neurons given that most dead neurons arise from their potentials largely biased toward the negative side.

BPLC with spike response model

Spike response model mapped onto computational graphs

We consider SRM, which is equivalent to the basic leaky integrate-and-fire (LIF) model with exponentially decaying synaptic current [13]. But our model is allowed to maximally spike only once in response to a single input sample by using an infinite hard refractory period in place of the refractory kernel. The choice of SRM, rather than simpler models, e.g., Stein's model [35], is to enlarge the mutual information of spike timing and synaptic weight, which is the key to temporal code.

In SRM, the subthreshold potential of the ith spiking neuron in the lth layer ($u_i^{(l)}$) is given by

$$u_i^{(l)}[t] = \sum_j w_{ij}^{(l)} \left(\epsilon * s_j^{(l-1)}\right)[t]\, sav_i^{(l)}[t], \tag{3}$$

where j denotes the indices of the presynaptic neurons, and $w_{ij}^{(l)}$ denotes the synaptic weight from the jth neuron in the (l-1)th layer. The spiking-availability function $sav_i^{(l)}$ is

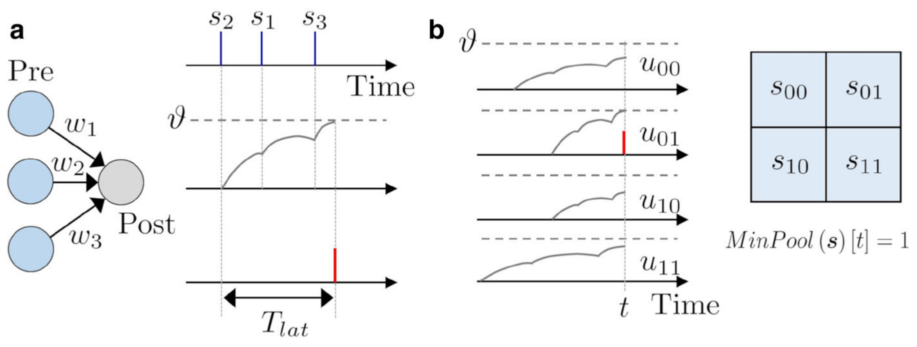

Fig. 1 **a** Definition of spiking latency, **b** Schematic of the minimum latency pooling operation

employed to allow each neuron to spike once at most such that $sav_i^{(l)} = 1$ if the neuron has not spiked before, and $sav_i^{(l)} = 0$ otherwise. The kernel ϵ is expressed as follows [13].

$$\epsilon = \frac{\tau_m}{\tau_m - \tau_s}\left(e^{-t/\tau_m} - e^{-t/\tau_s}\right)\Theta[t], \tag{4}$$

where Θ denotes the Heaviside step function. The potential and synaptic current time constants are denoted by τ_m and τ_s, respectively. A spike from the jth neuron in the (l-1)th layer at $\hat{t}_j^{(l-1)}$ is denoted by $s_j^{(l-1)}$. Because the kernel in Eq. (4) consists of two independent sub-kernels,

$$\epsilon_{(\cdot)} = \frac{\tau_m}{\tau_m - \tau_s} e^{-t/\tau_{(\cdot)}}\Theta[t], \text{ where } (\cdot) \in \{m, s\}, \tag{5}$$

Eq. (3) can be expressed as

$$\begin{aligned} u_i^{(l)}[t] &= \left(u_{i,m}^{(l)}[t] - u_{i,s}^{(l)}[t]\right) sav_i^{(l)}[t], \\ u_{i,(\cdot)}^{(l)}[t] &= \sum_j \frac{\tau_m w_{ij}^{(l)}}{\tau_m - \tau_s} e^{-\left(t-\hat{t}_j^{(l-1)}\right)/\tau_{(\cdot)}}\Theta\left[t - \hat{t}_j^{(l-1)}\right], \\ &\quad \text{where } (\cdot) \in \{m, s\}. \end{aligned}$$

Here, we introduce a new variable $v_j^{(l)}$ given by

$$\begin{aligned} v_j^{(l)}[t] &= v_{j,m}^{(l)}[t] - v_{j,s}^{(l)}[t], \\ v_{j,(\cdot)}^{(l)}[t] &= \frac{\tau_m}{\tau_m - \tau_s} e^{-\left(t-\hat{t}_j^{(l-1)}\right)/\tau_{(\cdot)}}\Theta\left[t - \hat{t}_j^{(l-1)}\right], \\ &\quad \text{where } (\cdot) \in \{m, s\}. \end{aligned}$$

The variables $u_{i,m}^{(l)}$ and $u_{i,s}^{(l)}$ are reset to zero when the neuron fires a spike. The advantage of this method is that the membrane potential can be evaluated by simply convolving input spikes using two independent kernels, which otherwise needs to solve two sequential differential equations [20]. After spiking, the spiking-availability function $sav_i^{(l)}$ remains constant at zero, hindering additional spike generation.

All variables are recursively evaluated using the explicit finite difference method.

$$\begin{aligned} v_{j,(\cdot)}^{(l)}[t+1] &= v_{j,(\cdot)}^{(l)}[t]\, e^{-1/\tau_{(\cdot)}} + \frac{\tau_m}{\tau_m - \tau_s} s_j^{(l-1)}[t+1], \\ &\quad \text{where } (\cdot) \in \{m, s\}, \\ u_{i,(\cdot)}^{(l)}[t+1] &= \sum_j w_{ij}^{(l)} v_{j,(\cdot)}^{(l)}[t+1], \text{ where } (\cdot) \in \{m, s\}, \\ u_i^{(l)}[t+1] &= \left(u_{i,m}^{(l)}[t+1] - u_{i,s}^{(l)}[t+1]\right) sav_i^{(l)}[t+1]. \end{aligned} \tag{6}$$

Equation (6) can be mapped onto a computational graph as shown in Fig. 2. A layer's processed data is transmitted along the forward pass through the use of spikes ($\boldsymbol{s}^{(l)}$).

Backward pass and gradients

SNNs are typically trained using forward and backward passes aligned in opposing directions, so that it is unavoidable to use surrogate gradients due to non-differentiability of spikes [29,34,44]. Instead, BPLC+NOSO uses a backward pass via spike timings $\hat{\boldsymbol{t}}^{(\cdot)}$ rather than spikes themselves $\boldsymbol{s}^{(\cdot)}$ (Fig. 2). This backward pass involves differentiable functions only. The output of NOSONet (with M output NOSOs) is the spiking latency values of the output NOSOs, $\boldsymbol{T}_{\text{lat}}^{(L)} = \{T_{\text{lat},i}^{(L)}\}_{i=1}^{M}$, as given in Eq. (1). The prediction is then made by reference to the output neuron of the minimum spiking latency. We use a cross-entropy loss function $\mathcal{L}(-\boldsymbol{T}_{\text{lat}}^{(L)}, \hat{\boldsymbol{y}})$, where $\hat{\boldsymbol{y}}$ denotes a one-hot encoded label vector. The loss is evaluated at the end of the learning phase, and the weights are then updated using the gradients assessed when the neurons spiked.

We calculate the weight's update $\Delta w_{ij}^{(l)}$ using the gradient descent method as follows.

$$\begin{aligned} \Delta w_{ij}^{(l)} &= -\eta \frac{\partial \mathcal{L}}{\partial \hat{t}_i^{(l)}} \frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}} \frac{\partial u_i^{(l)}}{\partial w_{ij}^{(l)}} \left[\hat{t}_i^{(l)}\right] \\ &= -\eta \frac{\partial \mathcal{L}}{\partial \hat{t}_i^{(l)}} \frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}} v_j^{(l)} \left[\hat{t}_i^{(l)}\right]. \end{aligned} \tag{7}$$

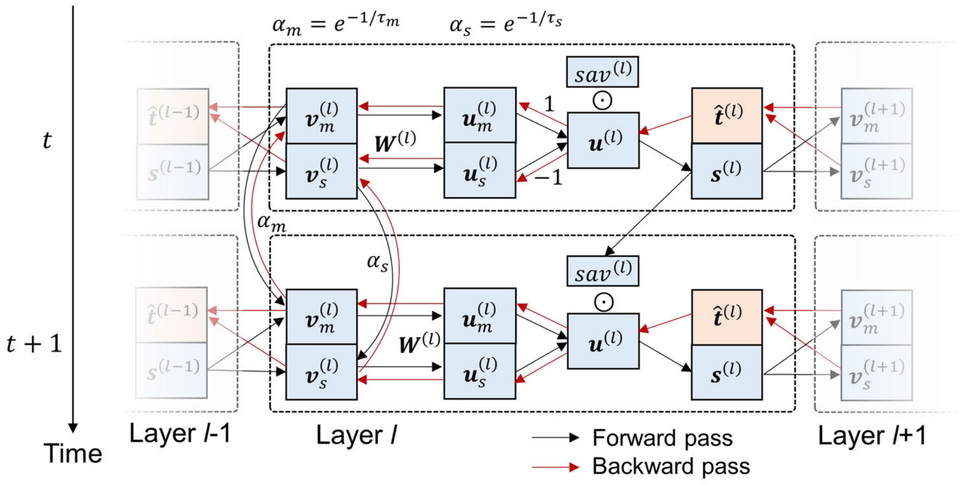

Fig. 2 Unfolded NOSONet on a computational graph

The learning rate and loss function are denoted by η and $\mathcal{L}$, respectively. Equation (7) is equivalent to

$$\Delta \boldsymbol{w}^{(l)} = -\eta diag\left(\boldsymbol{e}^{(l)}\right) \boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right], \tag{8}$$

with the error $\boldsymbol{e}^{(l)}$ given by

$$\boldsymbol{e}^{(l)} = \nabla_{\hat{\boldsymbol{t}}^{(l)}}\mathcal{L} \odot \hat{\boldsymbol{t}}^{(l)\prime},$$
$$\nabla_{\hat{\boldsymbol{t}}^{(l)}}\mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \hat{t}_i^{(l)}}, \cdots, \frac{\partial \mathcal{L}}{\partial \hat{t}_N^{(l)}}\right]^{\mathrm{T}},$$
$$\hat{\boldsymbol{t}}^{(l)\prime} = \left[\frac{\partial \hat{t}_1^{(l)}}{\partial u_1^{(l)}}, \cdots, \frac{\partial \hat{t}_N^{(l)}}{\partial u_N^{(l)}}\right]^{\mathrm{T}}, \tag{9}$$

for N neurons in the lth layer. The symbol $\odot$ denotes the Hadamard product. The matrix $\boldsymbol{v}^{(l)}[\hat{\boldsymbol{t}}^{(l)}]$ is given by

$$\boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right] = \begin{bmatrix} v_1^{(l)}\left[\hat{t}_1^{(l)}\right] & \cdots & v_M^{(l)}\left[\hat{t}_1^{(l)}\right] \\ \vdots & \ddots & \vdots \\ v_1^{(l)}\left[\hat{t}_N^{(l)}\right] & \cdots & v_M^{(l)}\left[\hat{t}_N^{(l)}\right] \end{bmatrix},$$

for M neurons in the $(l-1)$th layer.

The backward propagation of the error from the lth layer to the $(l-1)$th layer (with M neurons) is given by

$$\boldsymbol{e}^{(l-1)} = \left(\boldsymbol{w}^{(l)\mathrm{T}} \odot \boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]\right)\boldsymbol{e}^{(l)} \odot \hat{\boldsymbol{t}}^{(l-1)\prime},$$
$$\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right] = \begin{bmatrix} \frac{\partial v_1^{(l)}}{\partial \hat{t}_1^{(l-1)}}\left[\hat{t}_1^{(l)}\right] & \cdots & \frac{\partial v_1^{(l)}}{\partial \hat{t}_1^{(l-1)}}\left[\hat{t}_N^{(l)}\right] \\ \vdots & \ddots & \vdots \\ \frac{\partial v_M^{(l)}}{\partial \hat{t}_M^{(l-1)}}\left[\hat{t}_1^{(l)}\right] & \cdots & \frac{\partial v_M^{(l)}}{\partial \hat{t}_M^{(l-1)}}\left[\hat{t}_N^{(l)}\right] \end{bmatrix}. \tag{10}$$

Equation (10) is derived in Appendix A. Because NOSO spikes once at most, the elements once written in $\boldsymbol{v}^{(l)}[\hat{\boldsymbol{t}}^{(l)}]$ and $\boldsymbol{v}^{(l)\prime}[\hat{\boldsymbol{t}}^{(l)}]$ are not overwritten. Equation (10) identifies that BPLC involves the gradients of spike timings rather than spikes themselves. Therefore, the backward pass differs from the forward pass.

Two types of gradients are thus required for BPLC + NOSO: (i) $\partial \hat{t}_i^{(l)}/\partial u_i^{(l)}$ and (ii) $\partial v_j^{(l)}/\hat{t}_j^{(l-1)}$ at the spike timing $\hat{t}_i^{(l)}$. Fortunately, SRM allows these gradients to be expressed analytically.

Theorem 1 *When an SRM neuron (whose membrane potential is $u_i^{(l)}$) spikes at a given time $t (= \hat{t}_i^{(l)})$, the gradient of spike timing $\hat{t}_i^{(l)}$ with membrane potential is given by*

$$\frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}} = \left(u_{i,m}^{(l)}\left[\hat{t}_i\right]/\tau_m - u_{i,s}^{(l)}\left[\hat{t}_i\right]/\tau_s\right)^{-1}. \tag{11}$$

The proof of Theorem 1 is given in Appendix B. If the neuron does not spike during a learning phase, the gradient in Eq. (11) is zero.

Theorem 2 *When an SRM neuron receives an input spike at $\hat{t}_j^{(l-1)}$, the gradients of $v_{j,m}^{(l)}$ and $v_{j,s}^{(l)}$ with respect to $\hat{t}_j^{(l-1)}$ are given by*

$$\frac{\partial v_{j,(\cdot)}^{(l)}}{\partial \hat{t}_j^{(l-1)}}[t] = \frac{\tau_m}{\tau_{(\cdot)}\left(\tau_m - \tau_s\right)} e^{-\left(t-\hat{t}_j^{(l-1)}\right)/\tau_{(\cdot)}} \Theta\left[t - \hat{t}_j^{(l-1)}\right]$$
$$= \frac{v_{j,(\cdot)}^{(l)}[t]}{\tau_{(\cdot)}}, \text{ where } (\cdot) \in \{m, s\}. \tag{12}$$

Table 2 Classification accuracy and the number of spikes used for inference

Method	Network	Coding	Best accuracy	Average Accuracy	#spikes N_{sp}
Fashion-MNIST					
Ikegawa et al. [16]	16C3-{32C3}*6-{64C3}*5	Rate	89.10	–	7156K
Zhang et al. [46]	16C5-P2-32C5-P2-800-128	Temporal	90.10	–	–
Zhang et al. [47]	400-R400	Rate	90.13	90.00±0.14	–
Sun et al. [36]	32C3-P2-32C3-P2-128	Rate	91.56	–	12K (only Conv)
Cheng et al. [6]	32C3-P2-32C3-P2-128	Rate	92.07	–	–
Mirsadeghi et al. [25]	20C5-P2-40C5-P2-1000	Temporal	92.80	–	–
Zhang and Li. [48]	32C5-P2-64C5-P2-1024	Temporal	92.83	92.69±0.09	–
Zhao et al. [49]	32C5-P2-64C5-P2-1024	Rate	93.45	93.04±0.31	–
BPLC + NOSO	32C5-P2-64C5-P2-600	Latency	92.47	92.44±0.02	14K±0.26K
CIFAR-10					
Wu et al. [39]	CNN1*	Rate	85.24	–	–
Wu et al. [39]	CNN2**	Rate	90.53	–	–
Wu et al. [39]	CNN2-half-ch	Rate	87.80	–	1298K
Zhang and Li. [48]	CNN1	Temporal	89.22	–	–
Zhang and Li. [48]	CNN2	Temporal	91.41	–	308K
Tan et al. [37]	CNN1	Modified rate	89.57	–	412K
Tan et al. [37]	CNN2	Modified rate	90.13	–	342K
Zhao et al. [49]	CNN3***	Rate	90.93	–	–
Lee et al. [23]	ResNet11	Rate	90.95	–	1530K
BPLC + NOSO	CNN4****	Latency	89.77	89.37±0.25	142K±1.86K

*96C3-256C3-P2-384C3-P2-384C3-256C3-1024-1024
**128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512
***128C3-P2-256C3-P2-512C3-P2-1024
****64C5-128C5-P2-256C5-P2-512C5-256C5-1024-512

The proof of Theorem 2 is also given in Appendix B. Using Theorem 2, the gradient $\partial v_j^{(l)}/\hat{t}_j^{(l-1)}$ is given by

$$\frac{\partial v_j^{(l)}}{\partial \hat{t}_j^{(l-1)}}\left[\hat{t}_i^{(l)}\right] = v_{j,m}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_m - v_{j,s}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_s. \tag{13}$$

Likewise, this gradient is also zero if this neuron does not spike. Both gradients in Eqs. (11) and (13) can simply be calculated by reading out the four local variables ($u_{i,m}^{(l)}$, $u_{i,s}^{(l)}$, $v_{j,m}^{(l)}$, $v_{j,s}^{(l)}$) when the neuron spikes.

The above derivations are for folded NOSONet, where all tensors for each layer are simply overwritten over time so that the space complexity is independent of the number of timesteps. We used unfolded NOSONet in the temporal domain to apply the the automatic differentiation framework [31]. The equivalence between folded and unfolded NOSONets is proven in Appendix C.

Experiments

Convolutional NOSONet (C-NOSONet) was trained on Fashion-MNIST [40] and CIFAR-10 [22] using BPLC + NOSO. We used the hyperparameters listed in Appendix E unless otherwise stated. The hyperparameters were manually searched. All experiments were conducted in the Pytorch framework [31] on a GPU workstation (CPU: Intel Xeon Processor Gold, GPU: RTX A6000). NOSONet on Fashion-MNIST was trained using one GPU, whereas NOSONet on CIFAR-10 using four GPUs.

Classification accuracy and the number of spikes for inference

We evaluated the classification accuracy on Fashion-MNIST and CIFAR-10 and the total number of spikes used for inference $N_{\mathrm{sp}}(=\sum_{i,t} n_{\mathrm{sp}}^{(i,t)})$, where $n_{\mathrm{sp}}^{(i,t)}$ denotes the number of spikes generated from the layer i at timestep t.

Fashion-MNIST: Fashion-MNIST consists of 70,000 gray-scale images (each of which 28×28 in size) of clothing categorized as 10 classes [40]. We rescaled each gray-scale pixel value of an image to the range $0 - 0.3$ and

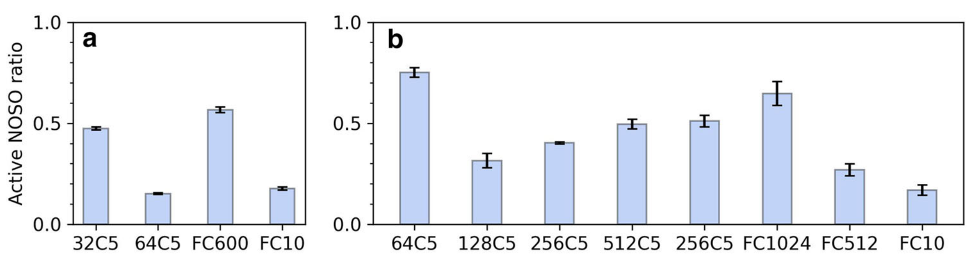

Fig. 3 Active NOSO ratio $\overline{n}_{sp}^{(i)}$ for each layer on **a** Fashion-MNIST and **b** CIFAR-10 over all timesteps

applied an additive white Gaussian noise (zero mean and 0.05 standard deviation). These values were then used as input currents into input LIF neurons. We trained a C-NOSONet (32C5-MP2-64C5-MP2-600, where MP denotes MinPool). The classification accuracy of the C-NOSONet is shown in Table 2 in comparison with previous works. We also evaluated the total number of spikes N_{sp} over all hidden+output NOSOs in the network for each test sample (Table 2). The results highlight large sparsity of active NOSOs, which likely reduces the inference latency when implemented in neuromorphic hardware. This will be discussed in Section "Discussion". Figure 3**a** shows the ratio of active NOSOs to all NOSOs, $\overline{n}_{\text{sp}}^{(i)} (= \sum_t n_{\text{sp}}^{(i,t)} / C^{(i)} H^{(i)} W^{(i)})$, for layer i over the entire timesteps.

CIFAR-10: CIFAR-10 consists of 60,000 real-world color images (each of which $3 \times 32 \times 32$ in size) of objects labeled as 10 classes [22]. All training images were preprocessed such that each image with zero-padding of size 4 was randomly cropped to 32×32, which was followed by random horizontal flipping. The RGB values of each pixel were rescaled to the range $0 - 0.3$ and then used as input currents. For learning stability, we linearly increased the initial learning rate (1E-2) to the plateau learning rate (5E-2) for the first five epochs (ramp rate: 8E-3/epoch). The fully trained C-NOSONet (64C5-128C5-MP2-256C5-MP2-512C5-256C5-1024-512) yields the classification accuracy and the number of spikes for inference in Table 2. Notably, our classification accuracy exceeds the result from an SNN of the same depth and width (CNN2-half-ch) [39] by approximately 2.0%. Additionally, our NOSONet uses much fewer spikes (only 10.9% of CNN2-half-ch), supporting high-throughput inference. The layer-wise active NOSO ratio $\overline{n}_{\text{sp}}^{(i)}$ over the entire timesteps is plotted in Fig. 3**b**, highlighting the high sparsity of spikes.

Minimum-latency pooling versus MaxPool

MinPool supports the latency code by passing the event of the minimum spiking latency in a given 2D patch. To identify its effects on learning, we compared NOSONets with MinPool layers and conventional MaxPool layers. Figures 4 and 5 show the comparisons on Fashion-MNIST and CIFAR-10, respectively. Compared with MaxPool, MinPool yields (i) the higher classification accuracy as shown in Figs. 4**a** and 5**a** and (ii) higher spike sparsity as shown in Figs. 4**c** and 5**c**. The accuracy increase despite the decrease in spike number may imply that MinPool removes unimportant spikes in classification unlike dropout that randomly removes spikes.

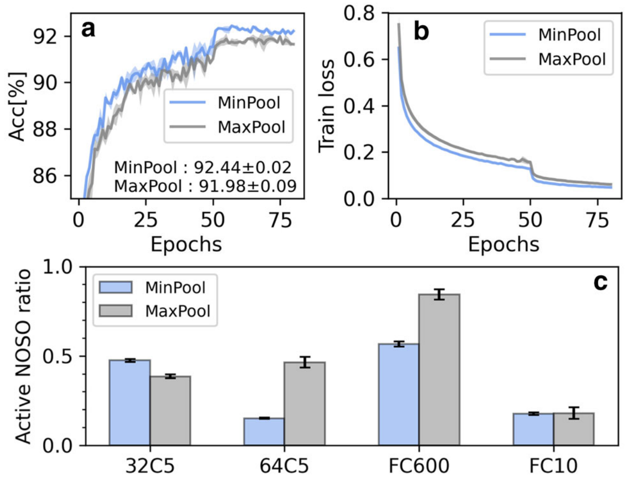

Fig. 4 Comparison between MinPool and MaxPool in terms of **a** validation accuracy, **b** training loss, and **c** layer-wise active NOSO ratio on Fashion-MNIST

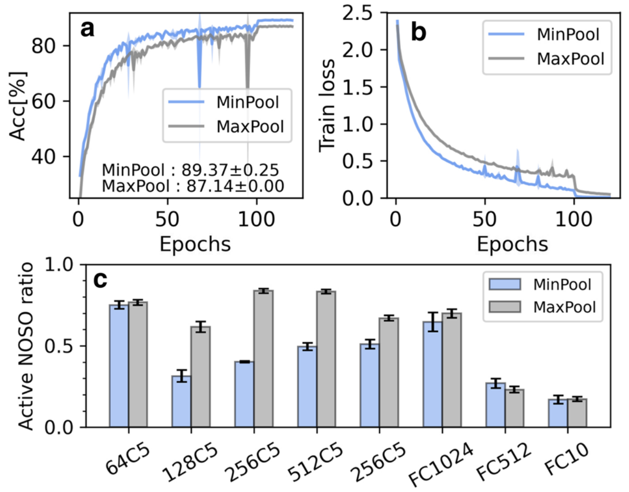

Fig. 5 Comparison between MinPool and MaxPool in terms of **a** validation accuracy, **b** training loss, and **c** layer-wise active NOSO ratio on CIFAR-10

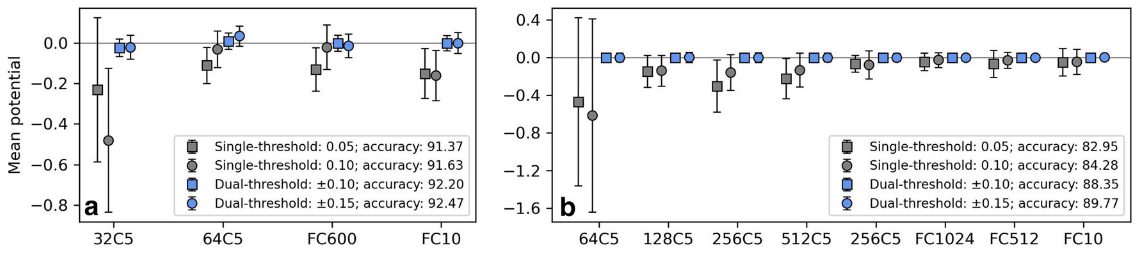

Fig. 6 Mean potential and standard deviation for neurons in each layer of NOSONet **a** on Fashion-MNIST, **b** on CIFAR-10. They were evaluated from potential distribution over samples in a random batch (size: 300 on Fashion-MNIST and 100 on CIFAR-10)

Effect of symmetric dual threshold on potential distribution

We identified the effect of the dual threshold on potential distribution over samples in a given batch by training NOSONet (32C5-MP2-64C5-MP2-600) on Fashion-MNIST and CIFAR-10 with four different threshold conditions: single threshold 0.05 and 0.1, and dual threshold ± 0.1 and ± 0.15. The results are shown in Fig. 6. The usage of dual threshold greatly lowers the standard deviation and results in a mean that is almost zero because it limits the potential to the range between $-\vartheta$ and ϑ. Additionally, the highest accuracy was attained with the dual threshold ± 0.15. The potential distributions for a single threshold case (0.1) and dual threshold case (± 0.15) on Fashion-MNIST are detailed in Appendix F.

Discussion

We estimate the inference time for an SNN mapped onto a general digital multicore neuromorphic processor using the following assumptions.

Assumption 1: Total N_n neurons in a given SNN are distributed uniformly over N_c cores of a neuromorphic processor, i.e., $N_\mathrm{n}/N_\mathrm{c}$ neurons per core.

Assumption 2: All $N_\mathrm{n}/N_\mathrm{c}$ neurons in each core share a multiplier by time-division multiplexing, so that the current potential is multiplied by a potential decay factor (e^{-1/τ_m}) for one neuron at each cycle.

Assumption 3: Synaptic operations are also executed serially.

Assumption 4: Neurons in different cores are updated parallel.

Each timestep for an SNN with LIF neurons includes two primary processes: (i) the process of multiplying the current potential by a decay factor and (ii) synaptic operation (spike routing to the destination neurons plus the consequent potential update). Process (i) in a digital neuromorphic processor is commonly pipelined within a core but executed in parallel over the N_c cores [20]. Thus, at each timestep, the time for process (i) for all N_n neurons (T_up) is given by

$$T_\mathrm{up} = (N_\mathrm{n}/N_\mathrm{c} + a)\, f_\mathrm{clk}^{-1},$$

where a and f_clk denote the initialization cycle number and clock speed, respectively. Although the number of initialization cycles a differs for different processor designs, it is commonly a few clock cycles. Given the total number of spikes generated at timestep t ($n_\mathrm{sp}[t]$), the time for synaptic operations at each timestep is given by

$$T_\mathrm{sop} = n_\mathrm{sp}[t]\, (\mathrm{SynOPS})^{-1}.$$

Given **Assumptions**, the total time for processes (i) and (ii) at each timestep is given by $T_\mathrm{step} = T_\mathrm{up} + T_\mathrm{sop}$. Therefore, we have the total time for inference during total N_step timesteps, $T_\mathrm{inf} = \sum_t T_\mathrm{step}[t]$, as follows.

$$T_\mathrm{inf} = N_\mathrm{step}\, (N_\mathrm{n}/N_\mathrm{c} + a)\, f_\mathrm{clk}^{-1} + N_\mathrm{sp}\, (\mathrm{SynOPS})^{-1}, \tag{14}$$

where $N_\mathrm{sp} = \sum_t n_\mathrm{sp}[t]$. The number of neurons in a core ($N_\mathrm{n}/N_\mathrm{c}$) differs for different designs. We assume 1k neurons in each core [8], a few tens MSynOPS as for [3,12,27], and 100 MHz clock speed. For inference involving N_{sp} spikes ($\sim 10^6$ as in Table 2) and a N_step of ~ 100, Eq. (14) identifies that T_sop is dominant over T_up so that T_inf is dictated by T_sop. Therefore, it is desired to concern N_sp when developing learning algorithms.

For SNNs with IF neurons (without leakage), process (i) is unnecessary so that T_up vanishes. Therefore, T_inf is solely determined by N_sp.

Conclusion and outlook

We proposed a mathematically rigorous learning algorithm (BPLC) based on spiking latency code in conjunction with minimum-latency pooling (MinPool) operations. We overcome the dead neuron issue using a symmetric dual threshold for spiking, which additionally improves the potential distribution over samples in a given batch (and thus the classification accuracy). BPLC-trained NOSONet on CIFAR-10 highlights its high accuracy outperforming the SNN of the same depth and width by approximately 2% with much fewer spikes (only 10.9%). This large reduction in the number of spikes largely reduces the inference latency of SNNs implemented in digital neuromorphic processors.

Currently, we conceive the following future work to boost the impact of BPLC + NOSO.

- **Scalability confirmation:** Although the viability of BPLC + NOSO was identified, its applicability to deeper SNNs on more complex datasets should be confirmed. Such datasets include not only static image datasets like ImageNet [33] but also event datasets like CIFAR10-DVS [24] and DVS128 Gesture [1]. Given that the number of spikes is severely capped, BPLC + NOSO on event datasets in particular might be challenging.
- **Hyperparameter fine-tuning:** To further increase the classification accuracy, the hyperparameters should be fine-tuned using optimization techniques.
- **Weight quantization:** BPLC + NOSO is based on full-precision (32b FP) weights. However, the viability of BPLC + NOSO with reduced precision weights should be confirmed to improve the efficiency in memory use. This may need an additional weight-quantization algorithm in conjunction with BPLC + NOSO like CBP [18].
- **Search for new application domains:** We need to search for new applications domains in which BPLC + NOSO can leverage its low process latency and power when implemented in neuromorphic hardware. The examples potentially include intelligent control systems like constrained nonlinear systems [41–43].

Author Contributions Conceptualization: Doo Seok Jeong, Dohun Kim, SeongMin Jin; Methodology: Dohun Kim, SeongMin Jin; Software: Doo Seok Jeong, Dohun Kim, SeongMin Jin, DongHyung Yoo; Investigation: Dohun Kim, SeongMin Jin, Jason Eshraghian; Writing-original draft: Doo Seok Jeong.

Funding This research was supported by National R&D Program through the National Research Foundation of Korea (NRF) funded by Ministry of Science and ICT (2021M3F3A2A01037632 and 2019R1C1C1009810).

Availability of data and materials The datasets generated during and/or analyzed during the current study are available in the GitHub repository, https://github.com/dooseokjeong/BPLC-NOSO.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Code availability The code is available in the GitHub repository, https://github.com/dooseokjeong/BPLC-NOSO.

Appendix A Derivation of backward propagation of errors

We define

$$\begin{aligned} C_\tau &= \frac{\tau_m}{\tau_m - \tau_s}, \\ S_j\left[t\right] &= \Theta\left[t - \hat{t}_j^{(l-1)}\right], \\ E_{j,(\cdot)}\left[t\right] &= e^{-\left(t-\hat{t}_j^{(l-1)}\right)/\tau_{(\cdot)}}, \text{ where } (\cdot) \in \{m, s\}. \end{aligned} \tag{A1}$$

The subthreshold membrane potential of NOSO is

$$u_i^{(l)}\left[t\right] = \sum_j C_\tau w_{ij}^{(l)} \left(E_{j,m}[t] - E_{j,s}[t]\right) S_j\left[t\right] sav_i^{(l)}\left[t\right]. \tag{A2}$$

Thus, the following equation holds when spiking with a spiking threshold ϑ.

$$\begin{aligned} \vartheta = &\sum_j C_\tau w_{ij}^{(l)} \left(E_{j,m}\left[\hat{t}_i^{(l)}\right] - E_{j,s}\left[\hat{t}_i^{(l)}\right]\right) \\ &\times S_j\left[\hat{t}_i^{(l)}\right] sav_i^{(l)}\left[\hat{t}_i^{(l)}\right]. \end{aligned} \tag{A3}$$

For simplicity, we omit the spiking-availability function $sav_i^{(l)}$ hereafter. The derivative $\partial\hat{t}_i^{(l)}/\partial\hat{t}_j^{(l-1)}$ is acquired by differentiating Eq. (A3) with respect to $\hat{t}_j^{(l-1)}$.

$$\begin{aligned} &\frac{\partial\hat{t}_i^{(l)}}{\partial\hat{t}_j^{(l-1)}} \\ &= \frac{C_\tau w_{ij}^{(l)} \left(\tau_m^{-1} E_{j,m}\left[\hat{t}_i^{(l)}\right] - \tau_s^{-1} E_{j,s}\left[\hat{t}_i^{(l)}\right]\right) S_j\left[\hat{t}_i^{(l)}\right]}{\sum_k C_\tau w_{ik}^{(l)} \left(\tau_m^{-1} E_{k,m}\left[\hat{t}_i^{(l)}\right] - \tau_s^{-1} E_{k,s}\left[\hat{t}_i^{(l)}\right]\right) S_k\left[\hat{t}_i^{(l)}\right]}. \end{aligned} \tag{A4}$$

According to Theorem 1, the denominator of the right-hand side of Eq. (A4) equals $\left(\partial\hat{t}_i^{(l)}/\partial u_i^{(l)}\right)^{-1}$, and thus we have

$$\frac{\partial\hat{t}_i^{(l)}}{\partial\hat{t}_j^{(l-1)}} = C_\tau w_{ij}^{(l)}\left(\frac{E_{j,m}\left[\hat{t}_i^{(l)}\right]}{\tau_m} - \frac{E_{j,s}\left[\hat{t}_i^{(l)}\right]}{\tau_s}\right) S_j\left[\hat{t}_i^{(l)}\right]\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}}. \tag{A5}$$

Applying a chain rule on the left-hand side of Eq. (A5) yields the following equation—

$$\frac{\partial u_i^{(l)}}{\partial\hat{t}_j^{(l-1)}} = C_\tau w_{ij}^{(l)}\left(\frac{E_{j,m}\left[\hat{t}_i^{(l)}\right]}{\tau_m} - \frac{E_{j,s}\left[\hat{t}_i^{(l)}\right]}{\tau_s}\right) S_j\left[\hat{t}_i^{(l)}\right]. \tag{A6}$$

Given that

$$v_{j,(\cdot)}^{(l)}[t] = C_\tau E_{j,(\cdot)}[t]\, S_j[t], \text{ where } (\cdot)\in\{m,s\}, \tag{A7}$$

Eq. (A6) is re-expressed as

$$\frac{\partial u_i^{(l)}}{\partial\hat{t}_j^{(l-1)}} = w_{ij}^{(l)}\left(v_{j,m}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_m - v_{j,s}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_s\right). \tag{A8}$$

According to Theorem 2,

$$\frac{\partial v_{j,(\cdot)}^{(l)}}{\partial\hat{t}_j^{(l-1)}}\left[\hat{t}_i^{(l)}\right] = \frac{C_\tau}{\tau_{(\cdot)}} E_{j,(\cdot)}\left[\hat{t}_i^{(l)}\right] S_j\left[\hat{t}_i^{(l)}\right],$$
$$\text{where } (\cdot)\in\{m,s\}.$$

Using Eq. (A7) at $t = \hat{t}_i^{(l)}$, the following equation holds: $\tau_{(\cdot)}^{-1} v_{j,(\cdot)}^{(l)}\left[\hat{t}_i^{(l)}\right] = \partial v_{j,(\cdot)}^{(l)}/\partial\hat{t}_j^{(l)}\left[\hat{t}_i^{(l)}\right]$, where $(\cdot)\in\{m,s\}$. Therefore, Eq. (A8) is re-arranged as

$$\frac{\partial u_i^{(l)}}{\partial\hat{t}_j^{(l-1)}} = w_{ij}^{(l)}\frac{\partial v_j^{(l)}}{\partial\hat{t}_j^{(l)}}\left[\hat{t}_i^{(l)}\right]. \tag{A9}$$

The error for the jth neuron in the $(l-1)$th layer $e_j^{(l-1)}$ is given by

$$\begin{aligned} e_j^{(l-1)} &= \frac{\partial\mathcal{L}}{\partial\hat{t}_j^{(l-1)}}\frac{\partial\hat{t}_j^{(l-1)}}{\partial u_j^{(l-1)}} \\ &= \sum_i\left(\frac{\partial\mathcal{L}}{\partial\hat{t}_i^{(l)}}\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}}\right)\frac{\partial u_i^{(l)}}{\partial\hat{t}_j^{(l-1)}}\frac{\partial\hat{t}_j^{(l-1)}}{\partial u_j^{(l-1)}} \\ &= \sum_i e_i^{(l)}\frac{\partial u_i^{(l)}}{\partial\hat{t}_j^{(l-1)}}\frac{\partial\hat{t}_j^{(l-1)}}{\partial u_j^{(l-1)}}. \end{aligned} \tag{A10}$$

Plugging Eq. (A9) into Eq. (A10) therefore leads to

$$e_j^{(l-1)} = \sum_i e_i^{(l)} w_{ij}^{(l)}\frac{\partial v_j^{(l)}}{\partial\hat{t}_j^{(l)}}\left[\hat{t}_i^{(l)}\right]\frac{\partial\hat{t}_j^{(l-1)}}{\partial u_j^{(l-1)}}. \tag{A11}$$

Equation (A11) is expressed as the following matrix formula.

$$\boldsymbol{e}^{(l-1)} = \left(\boldsymbol{w}^{(l)\mathrm{T}} \odot \boldsymbol{v}^{(l)'}\left[\hat{\boldsymbol{t}}^{(l)}\right]\right)\boldsymbol{e}^{(l)} \odot \hat{\boldsymbol{t}}^{(l-1)'},$$

where

$$\boldsymbol{v}^{(l)'}\left[\hat{\boldsymbol{t}}^{(l)}\right] = \begin{bmatrix} \frac{\partial v_1^{(l)}}{\partial\hat{t}_1^{(l-1)}}\left[\hat{t}_1^{(l)}\right] & \cdots & \frac{\partial v_1^{(l)}}{\partial\hat{t}_1^{(l-1)}}\left[\hat{t}_N^{(l)}\right] \\ \vdots & \ddots & \vdots \\ \frac{\partial v_M^{(l)}}{\partial\hat{t}_m^{(l-1)}}\left[\hat{t}_1^{(l)}\right] & \cdots & \frac{\partial v_M^{(l)}}{\partial\hat{t}_m^{(l-1)}}\left[\hat{t}_N^{(l)}\right] \end{bmatrix}.$$

Appendix B Proofs of Theorems

Theorem 1 When an SRM neuron (whose membrane potential is $u_i^{(l)}$) spikes at a given time $t(=\hat{t}_i^{(l)})$, the gradient of spike timing $\hat{t}_i^{(l)}$ with membrane potential is given by

$$\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}} = \left(u_{i,m}^{(l)}\left[\hat{t}_i\right]/\tau_m - u_{i,s}^{(l)}\left[\hat{t}_i\right]/\tau_s\right)^{-1}.$$

Proof The update of weight $w_{ij}^{(l)}$ is calculated using the gradient descent method as follows—

$$\begin{aligned} \Delta w_{ij}^{(l)} &= -\eta\frac{\partial\mathcal{L}}{\partial w_{ij}^{(l)}} \\ &= -\eta\frac{\partial\mathcal{L}}{\partial\hat{t}_i^{(l)}}\frac{\partial\hat{t}_i^{(l)}}{\partial w_{ij}^{(l)}} \\ &= -\eta\frac{\partial\mathcal{L}}{\partial\hat{t}_i^{(l)}}\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}}\frac{\partial u_i^{(l)}}{\partial w_{ij}^{(l)}}\left[\hat{t}_i^{(l)}\right]. \end{aligned} \tag{B12}$$

Regarding that $u_i^{(l)}[t] = w_{ij}^{(l)} v_j^{(l)}[t]$, the gradient $\partial u_i^{(l)}/\partial w_{ij}^{(l)}[t]$ equals $v_j^{(l)}[t]$. Consequently, we have

$$\Delta w_{ij}^{(l)} = -\eta\frac{\partial\mathcal{L}}{\partial\hat{t}_i^{(l)}}\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}} v_j^{(l)}\left[\hat{t}_i^{(l)}\right]. \tag{B13}$$

Differentiating Eq. (A3) with $w_{ij}^{(l)}$ yields

$$\frac{\partial \vartheta}{\partial w_{ij}^{(l)}} = v_j^{(l)}\left[\hat{t}_i^{(l)}\right] + \frac{\partial u_i^{(l)}}{\partial t}\left[\hat{t}_i^{(l)}\right]\frac{\partial \hat{t}_i^{(l)}}{\partial w_{ij}^{(l)}}. \tag{B14}$$

The left-hand side of Eq. (B14) is zero because the threshold ϑ is constant. Thus, the following equation holds—

$$\frac{\partial \hat{t}_i^{(l)}}{\partial w_{ij}^{(l)}} = -\left(\frac{\partial u_i^{(l)}}{\partial t}\left[\hat{t}_i^{(l)}\right]\right)^{-1} v_j^{(l)}\left[\hat{t}_i^{(l)}\right]. \tag{B15}$$

Plugging Eq. (B15) into Eq. (B12) yields

$$\Delta w_{ij}^{(l)} = \eta\frac{\partial \mathcal{L}}{\partial \hat{t}_i^{(l)}}\left(\frac{\partial u_i^{(l)}}{\partial t}\left[\hat{t}_i^{(l)}\right]\right)^{-1} v_j^{(l)}\left[\hat{t}_i^{(l)}\right]. \tag{B16}$$

A comparison between Eqs. (B13) and (B16) indicates that the following equation holds.

$$\frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}} = -\left(\frac{\partial u_i^{(l)}}{\partial t}\left[\hat{t}_i^{(l)}\right]\right)^{-1}. \tag{B17}$$

The right-hand side of Eq. (B17) is obtained by differentiating Eq. (A2) with t and evaluating the derivative at the spike timing $\hat{t}_i^{(l)}$, which finally leads to

$$\frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}} = \left(u_{i,m}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_m - u_{i,s}^{(l)}\left[\hat{t}_i^{(l)}\right]/\tau_s\right)^{-1},$$

where

$$u_{i,(\cdot)}^{(l)}\left[\hat{t}_i^{(l)}\right] = \sum_j C_\tau w_{ij}^{(l)} E_{j,(\cdot)}\left[\hat{t}_i^{(l)}\right] S_j\left[\hat{t}_i^{(l)}\right],$$
$$\text{where } (\cdot) \in \{m, s\}.$$

□

Theorem 2 When an SRM neuron receives an input spike at $\hat{t}_j^{(l-1)}$, the gradients of $v_{j,m}^{(l)}$ and $v_{j,s}^{(l)}$ with respect to $\hat{t}_j^{(l-1)}$ are given by

$$\frac{\partial v_{j,(\cdot)}^{(l)}}{\partial \hat{t}_j^{(l-1)}}[t] = \frac{C_\tau}{\tau_{(\cdot)}} E_{j,(\cdot)}[t]\, S_j[t], \text{ where } (\cdot) \in \{m, s\}.$$

Proof The variables $v_{j,m}^{(l)}$ and $v_{j,s}^{(l)}$ are given by

$$v_{j,(\cdot)}^{(l)}[t] = C_\tau E_{j,(\cdot)}[t]\, S_j[t], \text{ where } (\cdot) \in \{m, s\}. \tag{B18}$$

To be precise, the Heaviside step function in Eq. (B18) should be $\Theta\left[t - \hat{t}_j^{(l-1)} - \varepsilon\right]$ with $\varepsilon \to 0^+$ because $v_{j,(\cdot)}^{(l)}$ at $\hat{t}_j^{(l-1)}$ is $\tau_m/(\tau_m - \tau_s)$ rather than $\tau_m/[2(\tau_m - \tau_s)]$. Given this substitution, differentiating Eq. (B18) with respect to $\hat{t}_j^{(l-1)}$ yields

$$\frac{\partial v_{j,(\cdot)}^{(l)}}{\partial \hat{t}_j^{(l-1)}}[t] = \frac{C_\tau}{\tau_{(\cdot)}} E_{j,(\cdot)}[t]\, S_j[t], \text{ where } (\cdot) \in \{m, s\}.$$

□

Theorem 3 *Spike-stamp vectors for NOSOs satisfy the following equation:*

$$\boldsymbol{s}^{(l)}[t_1] \odot \boldsymbol{s}^{(l)}[t_2] = \begin{cases} \boldsymbol{s}^{(l)}[t_1] & \text{if } t_1 = t_2 \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \tag{B19}$$

Proof Because NOSOs spike once maximally, for all i, $s_i^{(l)}[t_1]\, s_i^{(l)}[t_2] = 0$ if $t_1 \neq t_2$, and $s_i^{(l)}[t_1]\, s_i^{(l)}[t_2] = s_i^{(l)}[t_1]$ if $t_1 = t_2$. Therefore, Eq. (B19) is true. □

Theorem 4 The weight update for the folded SNN,

$$\Delta \boldsymbol{w}^{(l)} = -\eta\, diag\left(\boldsymbol{e}^{(l)\mathrm{T}}\right) \boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right], \tag{B20}$$

is equivalent to the following equation—

$$\Delta \boldsymbol{w}^{(l)} = -\eta \sum_{t=1}^{T}\left(\bar{\boldsymbol{e}}^{(l)}[t] \odot \boldsymbol{s}^{(l)}[t]\right) \boldsymbol{v}^{(l)}[t]^{\mathrm{T}}, \tag{B21}$$

where $\boldsymbol{v}^{(l)}[t]$ is given by $\boldsymbol{v}^{(l)}[t] = \left[v_1^{(l)}[t], \cdots, v_m^{(l)}[t]\right]^{\mathrm{T}}$.

Proof The error $\boldsymbol{e}^{(l)}$ is known to be

$$\boldsymbol{e}^{(l)} = \sum_{t=1}^{T} \bar{\boldsymbol{e}}^{(l)}[t] \odot \boldsymbol{s}^{(l)}[t]. \tag{B22}$$

Using Eq. (B22) and a basic property of the Hamadard product, the matrix $diag\left(\boldsymbol{e}^{(l)\mathrm{T}}\right)$ on the right-hand side of Eq. (B20) is unfolded as

$$diag\left(\boldsymbol{e}^{(l)}\right) = \sum_{t=1}^{T} diag\left(\bar{\boldsymbol{e}}^{(l)}[t]\right) diag\left(\boldsymbol{s}^{(l)}[t]\right). \tag{B23}$$

The matrix $\boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right]$ in Eq. (B20), given by

$$\boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right] = \begin{bmatrix} v_1^{(l)}\left[\hat{t}_1^{(l)}\right] & \ldots & v_M^{(l)}\left[\hat{t}_1^{(l)}\right] \\ \vdots & \ddots & \vdots \\ v_1^{(l)}\left[\hat{t}_1^{(l)}\right] & \ldots & v_M^{(l)}\left[\hat{t}_N^{(l)}\right] \end{bmatrix},$$

is unfolded as

$$\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]=\sum_{t'=1}^{T}\boldsymbol{s}^{(l)}\left[t'\right]\boldsymbol{v}^{(l)}\left[t'\right]^{\mathrm{T}}. \tag{B24}$$

Entering Eqs. (B23) and (B24) into Eq. (B20) yields

$$\Delta\boldsymbol{w}^{(l)}=-\eta\sum_{t}^{T}\sum_{t'}^{T}\Big[diag\left(\bar{\boldsymbol{e}}^{(l)}\left[t\right]\right) diag\left(\boldsymbol{s}^{(l)}\left[t\right]\right)\boldsymbol{s}^{(l)}\left[t'\right]\boldsymbol{v}^{(l)}\left[t'\right]^{\mathrm{T}}\Big]. \tag{B25}$$

Note that $diag\left(\boldsymbol{s}^{(l)}\left[t\right]\right)\boldsymbol{s}^{(l)}\left[t'\right]=\boldsymbol{s}^{(l)}\left[t\right]\odot\boldsymbol{s}^{(l)}\left[t'\right]$, which is always zero if $t\neq t'$ according to Theorem 3. Therefore, we have

$$\begin{aligned}\Delta\boldsymbol{w}^{(l)}&=-\eta\sum_{t=1}^{T}diag\left(\bar{\boldsymbol{e}}^{(l)}\left[t\right]\right)\boldsymbol{s}^{(l)}\left[t\right]\boldsymbol{v}^{(l)}\left[t\right]^{\mathrm{T}}.\\&=-\eta\sum_{t=1}^{T}\left(\bar{\boldsymbol{e}}^{(l)}\left[t\right]\odot\boldsymbol{s}^{(l)}\left[t\right]\right)\boldsymbol{v}^{(l)}\left[t\right]^{\mathrm{T}}.\end{aligned}$$

□

Theorem 5 The backward propagation of errors

$$\boldsymbol{e}^{(l-1)}=\left(\boldsymbol{w}^{(l)\mathrm{T}}\odot\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]\right)\boldsymbol{e}^{(l)}\odot\hat{\boldsymbol{t}}^{(l-1)\prime} \tag{B26}$$

is unfolded over the timesteps as follows—

$$\begin{aligned}\boldsymbol{e}^{(l-1)}&=\sum_{t=1}^{T}\tilde{\boldsymbol{e}}^{(l-1)}[t],\\ \tilde{\boldsymbol{e}}^{(l-1)}[t]&=\sum_{t'=t}^{T}\left(\boldsymbol{w}^{(l)\mathrm{T}}\tilde{\boldsymbol{e}}^{(l)}[t']\right)\odot\boldsymbol{B}^{(l)}\left[t,t'\right]\\&\quad\odot\boldsymbol{A}^{(l-1)}\left[t\right]\odot\boldsymbol{s}^{(l-1)}\left[t\right],\\ \boldsymbol{B}^{(l)}\left[t,t'\right]&=C_{\tau}\left[\tau_m^{-1}e^{-(t'-t)/\tau_m}-\tau_s^{-1}e^{-(t'-t)/\tau_s}\right]\mathbf{1}.\end{aligned}$$

The all-one vector is denoted by $\mathbf{1}=[1,\cdots,1]^{\mathrm{T}}$.

Proof The matrix $\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]$ in Eq. (B26),

$$\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]=\begin{bmatrix}\frac{\partial v_1^{(l)}}{\partial\hat{t}_1^{(l-1)}}\left[\hat{t}_1^{(l)}\right]&\cdots&\frac{\partial v_1^{(l)}}{\partial\hat{t}_1^{(l-1)}}\left[\hat{t}_N^{(l)}\right]\\ \vdots&\ddots&\vdots\\ \frac{\partial v_M^{(l)}}{\partial\hat{t}_M^{(l-1)}}\left[\hat{t}_1^{(l)}\right]&\cdots&\frac{\partial v_M^{(l)}}{\partial\hat{t}_M^{(l-1)}}\left[\hat{t}_N^{(l)}\right]\end{bmatrix},$$

is unfolded as

$$\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]=\sum_{t=1}^{T}\boldsymbol{C}^{(l)}\left[t\right]\boldsymbol{s}^{(l)}\left[t\right]^{\mathrm{T}}. \tag{B27}$$

$$\boldsymbol{C}^{(l)}\left[t\right]=\left[\frac{\partial v_1^{(l)}}{\partial\hat{t}_1^{(l-1)}}\left[t\right],\cdots\frac{\partial v_M^{(l)}}{\partial\hat{t}_M^{(l-1)}}\left[t\right]\right]^{\mathrm{T}}. \tag{B28}$$

Its elements are given by

$$\frac{\partial v_j^{(l)}}{\partial\hat{t}_j^{(l-1)}}\left[t\right]=C_{\tau}\left(\frac{E_{j,m}\left[t\right]}{\tau_m}-\frac{E_{j,s}\left[t\right]}{\tau_s}\right)S_j\left[t\right]. \tag{B29}$$

Note that the element $\partial v_j^{(l)}/\partial\hat{t}_j^{(l-1)}\left[t\right]$ is a continuous function of $t\left(\geq\hat{t}_j^{(l-1)}\right)$, and $\boldsymbol{v}^{(l)\prime}\left[\hat{\boldsymbol{t}}^{(l)}\right]$ is the read of $\boldsymbol{A}^{(l)}\left[t\right]$ at $\hat{\boldsymbol{t}}^{(l)}$ using $\boldsymbol{s}^{(l)}$. Plugging Eqs. (B22) and (B27) into Eq. (B26) yields

$$\boldsymbol{e}^{(l-1)}=\sum_{t'=1}^{T}\sum_{t=1}^{T}\Big[\boldsymbol{w}^{(l)\mathrm{T}}\odot\left(\boldsymbol{A}^{(l)}\left[t'\right]\boldsymbol{s}^{(l)}\left[t'\right]^{\mathrm{T}}\right)\bar{\boldsymbol{e}}^{(l)}\left[t\right]\odot\boldsymbol{s}^{(l)}\left[t\right]\Big]\odot\hat{\boldsymbol{t}}^{(l-1)\prime}. \tag{B30}$$

We use a general property of the Hadamard product,

$$\left(\boldsymbol{w}\odot\boldsymbol{a}\boldsymbol{b}^{\mathrm{T}}\right)\boldsymbol{c}=\left[\boldsymbol{w}\left(\boldsymbol{b}\odot\boldsymbol{c}\right)\right]\odot\boldsymbol{a},$$

where $\boldsymbol{w}\in\mathbb{R}^{n\times m}$, $\boldsymbol{a}\in\mathbb{R}^{n}$, $\boldsymbol{b}\in\mathbb{R}^{m}$, and $\boldsymbol{c}\in\mathbb{R}^{m}$. Equation (B30) is consequently arranged as

$$\boldsymbol{e}^{(l-1)}=\sum_{t'=1}^{T}\sum_{t=1}^{T}\Big[\boldsymbol{w}^{(l)\mathrm{T}}\left(\bar{\boldsymbol{e}}^{(l)}\left[t\right]\odot\boldsymbol{s}^{(l)}\left[t'\right]\odot\boldsymbol{s}^{(l)}\left[t\right]\right)\odot\boldsymbol{C}^{(l)}\left[t'\right]\Big]\odot\hat{\boldsymbol{t}}^{(l-1)\prime}. \tag{B31}$$

Using Theorem 3, we have

$$\boldsymbol{e}^{(l-1)}=\sum_{t'=1}^{T}\Big[\boldsymbol{w}^{(l)\mathrm{T}}\left(\bar{\boldsymbol{e}}^{(l)}\left[t'\right]\odot\boldsymbol{s}^{(l)}\left[t'\right]\right)\odot\boldsymbol{C}^{(l)}\left[t'\right]\Big]\odot\hat{\boldsymbol{t}}^{(l-1)\prime}. \tag{B32}$$

Considering the following equations—

$$\begin{aligned}\frac{\partial\hat{t}_i^{(l)}}{\partial u_i^{(l)}}\left[t\right]&=A_i^{(l)}\left[t\right]\delta\left[t-\hat{t}_i^{(l)}\right],\\ A_i^{(l)}\left[t\right]&=\left(u_{i,m}^{(l)}\left[t\right]/\tau_m-u_{i,s}^{(l)}\left[t\right]/\tau_s\right)^{-1},\end{aligned}$$

The gradient $\hat{\boldsymbol{t}}^{(l-1)'}$ on the right-hand side of Eq. (B32) is unfolded as

$$\hat{\boldsymbol{t}}^{(l-1)'} = \sum_{t=1}^{T} \boldsymbol{A}^{(l-1)}[t] \odot \boldsymbol{s}^{(l-1)}[t]. \tag{B33}$$

From Eqs. (B32) and (B33), we have

$$\boldsymbol{e}^{(l-1)} = \sum_{t=1}^{T}\sum_{t'=t}^{T}\left[\boldsymbol{w}^{(l)\mathrm{T}}\left(\overline{\boldsymbol{e}}^{(l)}[t'] \odot \boldsymbol{s}^{(l)}[t']\right) \odot \boldsymbol{C}^{(l)}[t']\right] \odot \boldsymbol{A}^{(l-1)}[t] \odot \boldsymbol{s}^{(l-1)}[t]. \tag{B34}$$

Note that the lower limit of the summation over t' is set to t because $\boldsymbol{C}^{(l)}[t']$ in this equation becomes zero for any $t' < t$ according to Theorem 2 (see the Heaviside step function). As such, $\boldsymbol{C}^{(l)}[t'] = \left[\partial v_1^{(l)}/\partial \hat{t}_1^{(l-1)}[t'], \cdots \partial v_m^{(l)}/\partial \hat{t}_m^{(l-1)}[t']\right]^{\mathrm{T}}$. If we leave the presynaptic spike timing $\hat{t}_j^{(l-1)}$ as a variable t, the element becomes

$$\frac{\partial v_j^{(l)}}{\partial \hat{t}_j^{(l-1)}}[t'] = C_\tau\left[\tau_m^{-1}e^{-(t'-t)/\tau_m} - \tau_s^{-1}e^{-(t'-t)/\tau_s}\right]. \tag{B35}$$

As shown in Eq. (B34), the variable t is the time argument of the presynaptic spike-stamp vector $s^{(l-1)}$, so that t such that $s_j^{(l-1)}[t] = 1$ is $t_j^{(l-1)}$, rendering Eq. (B35) equal to Eq. (B29). For clearity, we introduce a new vector $\boldsymbol{B}^{(l)}[t,t']$ whose element is given by Eq. (B35). The product $\overline{\boldsymbol{e}}^{(l)}[t'] \odot \boldsymbol{s}^{(l)}[t']$ in Eq. (B33) is the error at the timestep t', i.e., $\tilde{\boldsymbol{e}}^{(l-1)}[t']$. Therefore, we eventually have

$$\boldsymbol{e}^{(l-1)} = \sum_{t=1}^{T}\tilde{\boldsymbol{e}}^{(l-1)}[t], \tag{B36}$$

where

$$\tilde{\boldsymbol{e}}^{(l-1)}[t] = \sum_{t_1=t}^{T}\left(\boldsymbol{w}^{(l)\mathrm{T}}\tilde{\boldsymbol{e}}^{(l-1)}[t']\right) \odot \boldsymbol{B}^{(l)}[t,t'] \odot \boldsymbol{A}^{(l-1)}[t] \odot \boldsymbol{s}^{(l-1)}[t], \tag{B37}$$

□

Appendix C Proof of equivalence between folded and unfolded NOSONets

NOSONet can be unfolded on a computational graph to use the the automatic differentiation framework [31]. To begin with, we define a spike-stamp vector at timestep t ($s^{(l)}[t]$) such that its element is '1' if the corresponding NOSO spikes at the timestep, and '0' otherwise.

$$\boldsymbol{s}^{(l)}[t] = \left[\delta\left[t - \hat{t}_1^{(l)}\right], \cdots, \delta\left[t - \hat{t}_n^{(l)}\right]\right]^{\mathrm{T}}.$$

Given that the variables $u_{i,m}^{(l)}$ and $u_{i,s}^{(l)}$ are continuous functions of time t, the gradient $\partial \hat{t}_i^{(l)}/\partial u_i^{(l)}$ in Eq. (11) is the read-out of $\left(u_{i,m}^{(l)}[t]/\tau_m - u_{i,s}^{(l)}[t]/\tau_s\right)^{-1}$ at $\hat{t}_i$. In this regard, Eq. (11) can be re-expressed as

$$\frac{\partial \hat{t}_i^{(l)}}{\partial u_i^{(l)}}[t] = A_i^{(l)}[t]\,\delta\left[t - \hat{t}_i^{(l)}\right],$$
$$A_i^{(l)}[t] = \left(u_{i,m}^{(l)}[t]/\tau_m - u_{i,s}^{(l)}[t]/\tau_s\right)^{-1}. \tag{C38}$$

Therefore, the error $\boldsymbol{e}^{(l)}$ in Eq. (9) is re-expressed as the read-out of the variable $\overline{\boldsymbol{e}}^{(l)}[t]$ (calculated at every timestep) upon spiking:

$$\boldsymbol{e}^{(l)} = \sum_{t=1}^{T}\overline{\boldsymbol{e}}^{(l)}[t] \odot \boldsymbol{s}^{(l)}[t],$$
$$\overline{\boldsymbol{e}}^{(l)}[t] = \nabla_{\hat{\boldsymbol{t}}^{(l)}}\mathcal{L} \odot \boldsymbol{A}^{(l)}[t]. \tag{C39}$$

Theorem 4 *The weight update for the folded SNN,*

$$\Delta\boldsymbol{w}^{(l)} = -\eta\, diag\left(\boldsymbol{e}^{(l)}\right)\boldsymbol{v}^{(l)}\left[\hat{\boldsymbol{t}}^{(l)}\right],$$

is equivalent to the following equation.

$$\Delta\boldsymbol{w}^{(l)} = -\eta\sum_{t=1}^{T}\left(\overline{\boldsymbol{e}}^{(l)}[t] \odot \boldsymbol{s}^{(l)}[t]\right)\boldsymbol{v}^{(l)}[t]^{\mathrm{T}},$$
$$\boldsymbol{v}^{(l)}[t] = \left[v_1^{(l)}[t], \ldots, v_M^{(l)}[t]\right]^{\mathrm{T}}. \tag{C40}$$

Theorem 5 *The backward propagation of errors in aggregate*

$$\boldsymbol{e}^{(l-1)} = \left(\boldsymbol{w}^{(l)\mathrm{T}} \odot \boldsymbol{v}^{(l)'}\left[\hat{\boldsymbol{t}}^{(l)}\right]\right)\boldsymbol{e}^{(l)} \odot \hat{\boldsymbol{t}}^{(l-1)'}$$

is decomposed into timestep-wise errors $\tilde{\boldsymbol{e}}^{(l-1)}[t]$, each of which is calculated at every timestep, as follows:

$$\boldsymbol{e}^{(l-1)} = \sum_{t=1}^{T}\tilde{\boldsymbol{e}}^{(l-1)}[t],$$
$$\tilde{\boldsymbol{e}}^{(l-1)}[t] = \sum_{t'=t}^{T}\left(\boldsymbol{w}^{(l)\mathrm{T}}\tilde{\boldsymbol{e}}^{(l)}[t']\right) \odot \boldsymbol{B}^{(l)}[t,t'] \odot \boldsymbol{A}^{(l-1)}[t] \odot \boldsymbol{s}^{(l-1)}[t],$$

$$\boldsymbol{B}^{(l)}\left[t, t'\right] = C_\tau \left[\tau_m^{-1} e^{-(t'-t)/\tau_m} - \tau_s^{-1} e^{-(t'-t)/\tau_s}\right] \mathbf{1}.$$

The all-one vector is denoted by $\mathbf{1} = [1, \ldots, 1]^{\mathrm{T}}$.

Theorems 4 and 5 are proven in Appendix B. Theorem 4 identifies the backward propagation of errors at timestep t' toward timestep t through time. Thus, BPLC+NOSO can be unfolded on a computational graph as shown in Fig. 2, allowing the automatic differentiation framework to be used to learn the weights. Note that we rule out the backward pass from $sav^{(l)}[t+1]$ to $\boldsymbol{s}^{(l)}[t]$ because it can be ignored if the learning uses spike function gradients (rather than surrogate gradients) and refractory periods. This is proven in Appendix D.

Appendix D Gradient of the spike-availability function with respect to a spike from the previous timestep

Spike-function gradients are non-zero only when the neurons spike unlike surrogate gradients. The same neuron cannot spike at the consecutive timesteps in a row because of the refractory period. Consider the computational graph in Fig. 2. When the ith neuron in the lth layer is quiet at timestep $t+1$, the gradient $\partial \hat{t}_i^{(l)} / \partial u_i^{(l)}[t+1]$ is zero, so that no gradient flows to $s_i^{(l)}[t]$ regardless of the presence of the backward pass. When the neuron is active at timestep $t+1$ (i.e., quiet at timestep t), the gradient $\partial \hat{t}_i^{(l)} / \partial u_i^{(l)}[t+1]$ is non-zero. However, the gradient at timestep t is zero, so that the presence or absence of the backward pass does not affect any gradient flow.

Appendix E Hyperparameters

We used the hyperparameters in Table 3. The input scaling factor is an upper limit of the scaled pixel value of input image. We initialized the kernels and weight matrices using the Xavier uniform initialization method given by

$$W \sim U\left(-\sqrt{\frac{a}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{a}{n_{\text{in}} + n_{\text{out}}}}\right),$$

where a is set to 6. The parameters in NOSONet (32C5-MP2-64C5-MP2-600) on Fashion-MNIST were initialized using the Xavier uniform method. We also initialized NOSONet (64C5-128C5-MP2-256C5-MP2-512C5-256C5-1024-512) on CIFAR-10 using the Xavier uniform method, but the weight matrices for the fully connected layers were initialized using a modified Xavier uniform method with $a = 3$ rather than 6.

Table 3 Hyperparameters used

Parameter	Value	
	F-MNIST	CIFAR-10
Timestep	1 ms	1 ms
Spiking threshold ϑ	±0.15 mV	± 0.15 mV
Optimizer	SGD	SGD
Input scaling factor	0.3	0.3
Membrane potential time constant τ_m	160 ms	165 ms
Synaptic current time constant τ_s	40 ms	50 ms
# Epochs	80	120
Batch size	64	32
Initial learning rate	5E-3	1E-2
Plateau learning rate	–	5E-2
Learning rate decay	0.1	0.1
Decay interval	50 epochs	100 epochs
Weight decay rate (L2 regularization)	5E-3	1E-3
# Timesteps	100	100
Initialization	Xavier uniform [14]	Xavier uniform [14]

Appendix F Potential distribution over samples in a batch

Figures 7 and 8 show potential distributions over samples in a random batch (batch size: 300) for single threshold and dual threshold cases, respectively. Note that the distributions exclude zero potential.

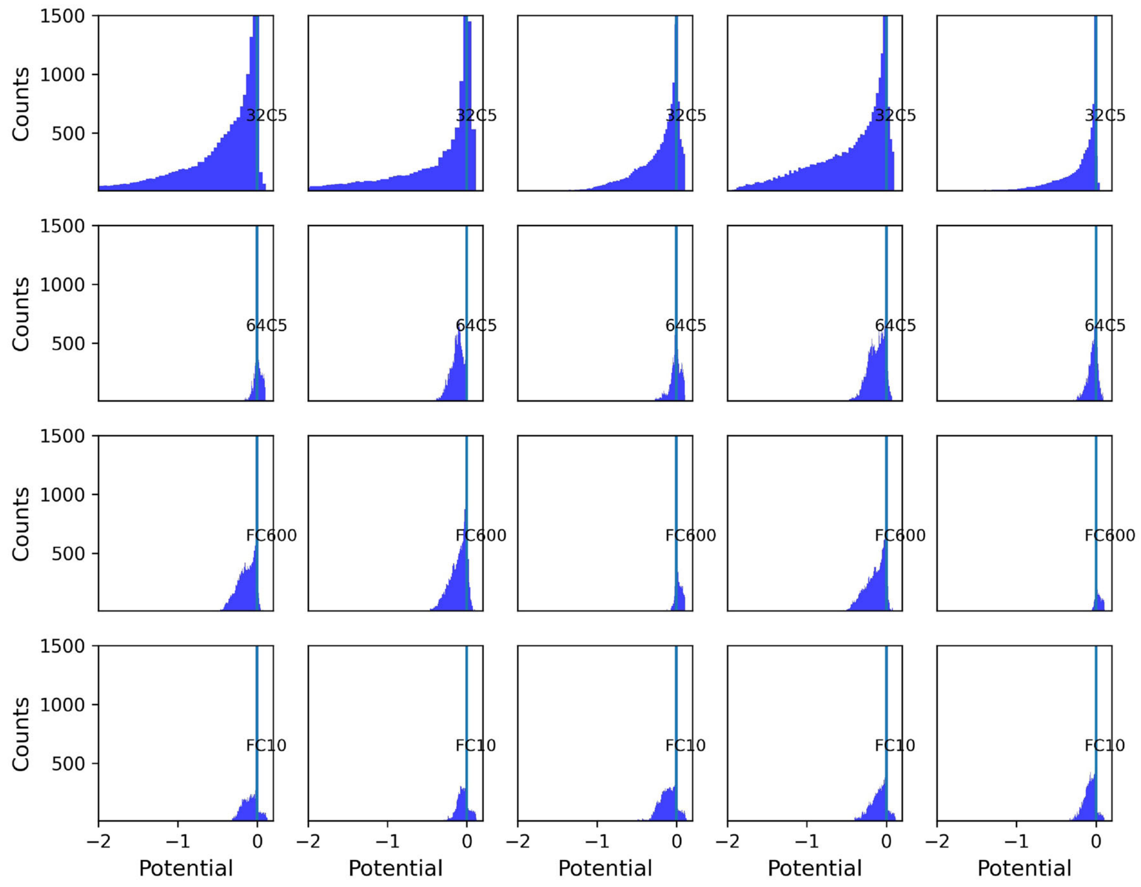

Fig. 7 Potential distribution over samples in a random batch (size: 300) for single threshold NOSOs ($\vartheta = 0.1$)

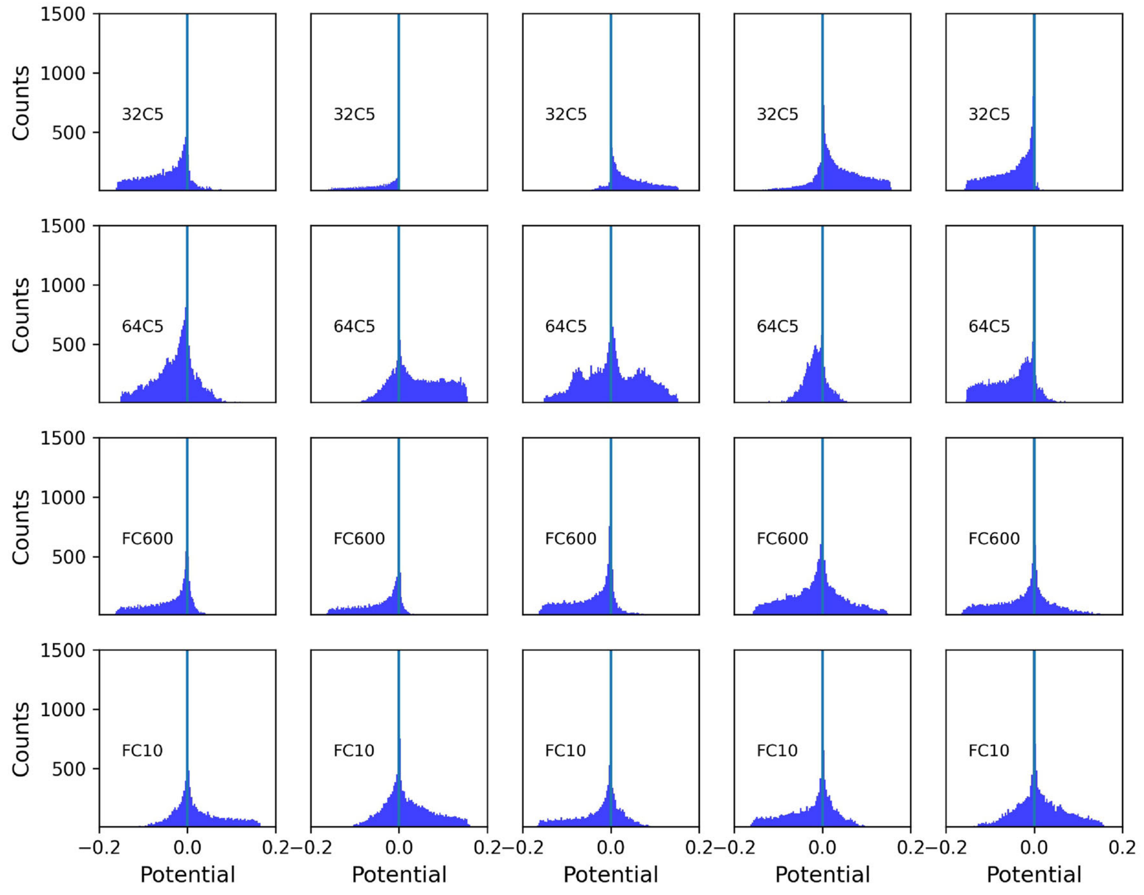

Fig. 8 Potential distribution over samples in a random batch (size: 300) for dual threshold NOSOs ($\vartheta = \pm 0.15$)

References

1. Amir A, Taba B, Berg D, et al (2017) A low power, fully event-based gesture recognition system. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 7243–7252. https://doi.org/10.1109/CVPR.2017.781
2. Bellec G, Salaj D, Subramoney A, et al (2018) Long short-term memory and learning-to-learn in networks of spiking neurons. In: Advances in Neural Information Processing Systems, vol 31. Curran Associates, Inc
3. Benjamin BV, Gao P, McQuinn E et al (2014) Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. Proc IEEE 102(5):699–716. https://doi.org/10.1109/JPROC.2014.2313565
4. Bi GQ, Poo MM (1998) Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. J Neurosci 18(24):10,464–10,472. https://doi.org/10.1523/JNEUROSCI.18-24-10464.1998
5. Bohte SM, Kok JN, La Poutre H (2002) Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48(1–4):17–37. https://doi.org/10.1016/S0925-2312(01)00658-0
6. Cheng X, Hao Y, Xu J, et al (2020) Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition. In: IJCAI, pp 1519–1525. https://doi.org/10.24963/ijcai.2020/211
7. Comsa IM, Potempa K, Versari L, et al (2020) Temporal coding in spiking neural networks with alpha synaptic function. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp 8529–8533, https://doi.org/10.1109/icassp40776.2020.9053856
8. Davies M, Srinivasa N, Lin TH et al (2018) Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro 38(1):82–99. https://doi.org/10.1109/MM.2018.112130359
9. Davies M, Wild A, Orchard G et al (2021) Advancing neuromorphic computing with loihi: a survey of results and outlook. Proc IEEE 109(5):911–934. https://doi.org/10.1109/JPROC.2021.3067593
10. Eshraghian JK, Ward M, Neftci E et al (2021) Training spiking neural networks using lessons from deep learning. https://doi.org/10.48550/ARXIV.2109.12894
11. Fang W, Yu Z, Chen Y et al (2021) Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 2661–2671. https://doi.org/10.1109/ICCV48922.2021.00266
12. Frenkel C, Lefebvre M, Legat JD et al (2018) A 0.086-mm ^{2} 1.27-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. IEEE Trans Biomed Circ Syst 13(1):145–158. https://doi.org/10.1109/TBCAS.2018.2880425
13. Gerstner W, Kistler WM (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge University Press, Cambridge
14. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth

international conference on artificial intelligence and statistics, pp 249–256

15. Hebb DO (1949) The organization of behavior. Wiley, New York
16. Si I, Saiin R, Sawada Y et al (2022) Rethinking the role of normalization and residual blocks for spiking neural networks. Sensors 22(8):2876. https://doi.org/10.3390/s22082876
17. Kheradpisheh SR, Masquelier T (2020) Temporal backpropagation for spiking neural networks with one spike per neuron. Int J Neural Syst 30(6):2050,027. https://doi.org/10.1142/S0129065720500276
18. Kim G, Jeong DS (2021) Cbp: backpropagation with constraint on weight precision using a pseudo-lagrange multiplier method. In: Advances in Neural Information Processing Systems, vol 34. Curran Associates, Inc
19. Kim J, Kim K, Kim JJ (2020a) Unifying activation-and timing-based learning rules for spiking neural networks. In: Advances in Neural Information Processing Systems, vol 33. Curran Associates, Inc
20. Kim J, Kornijcuk V, Ye C et al (2020) Hardware-efficient emulation of leaky integrate-and-fire model using template-scaling-based exponential function approximation. IEEE Trans Circ Syst I Regul Pap 68(1):350–362. https://doi.org/10.1109/TCSI.2020.3027583
21. Kornijcuk V, Kim D, Kim G et al (2020) Simplified calcium signaling cascade for synaptic plasticity. Neural Netw 123:38–51. https://doi.org/10.1016/j.neunet.2019.11.022
22. Krizhevsky A (2009) Learning multiple layers of features from tiny images https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
23. Lee C, Sarwar SS, Panda P et al (2020) Enabling spike-based backpropagation for training deep neural network architectures. Front Neurosci 14:119. https://doi.org/10.3389/fnins.2020.00119
24. Li H, Liu H, Ji X et al (2017) Cifar10-dvs: an event-stream dataset for object classification. Front Neurosci 11:309. https://doi.org/10.3389/fnins.2017.00309
25. Mirsadeghi M, Shalchian M, Kheradpisheh SR, et al (2021a) Spike time displacement based error backpropagation in convolutional spiking neural networks. arXiv preprint . https://doi.org/10.48550/arXiv.2108.13621
26. Mirsadeghi M, Shalchian M, Kheradpisheh SR et al (2021) Stidi-bp: spike time displacement based error backpropagation in multilayer spiking neural networks. Neurocomputing 427:131–140. https://doi.org/10.1016/j.neucom.2020.11.052
27. Moradi S, Qiao N, Stefanini F et al (2018) A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). IEEE Trans Biomed Circuits Syst 12(1):106–122. https://doi.org/10.1109/TBCAS.2017.2759700
28. Neckar A, Fok S, Benjamin BV et al (2019) Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model. Proc IEEE 107(1):144–164. https://doi.org/10.1109/JPROC.2018.2881432
29. Neftci EO, Mostafa H, Zenke F (2019) Surrogate gradient learning in spiking neural networks: bringing the power of gradient-based optimization to spiking neural networks. IEEE Signal Process Mag 36(6):51–63. https://doi.org/10.1109/MSP.2019.2931595
30. Park S, Kim S, Na B et al (2020) T2fsnn: Deep spiking neural networks with time-to-first-spike coding. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), pp 1–6. https://doi.org/10.1109/DAC18072.2020.9218689
31. Paszke A, Gross S, Massa F et al (2019) Pytorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems, vol 32. Curran Associates, Inc
32. Pfeiffer M, Pfeil T (2018) Deep learning with spiking neurons: opportunities and challenges. Front Neurosci 12:774. https://doi.org/10.3389/fnins.2018.00774
33. Russakovsky O, Deng J, Su H et al (2015) Imagenet large scale visual recognition challenge. Int J Comput Vision 115(3):211–252. https://doi.org/10.1007/s11263-015-0816-y
34. Shrestha SB, Orchard G (2018) Slayer: Spike layer error reassignment in time. In: Advances in Neural Information Processing Systems, vol 31. Curran Associates, Inc
35. Stein RB (1965) A theoretical analysis of neuronal variability. Biophys J 5(2):173–194. https://doi.org/10.1016/S0006-3495(65)86709-1
36. Sun C, Chen Q, Fu Y, et al (2022) Deep spiking neural network with ternary spikes. In: 2022 IEEE Biomedical Circuits and Systems Conference (BioCAS), pp 251–254. https://doi.org/10.1109/BioCAS54905.2022.9948581
37. Tan PY, Wu CW, Lu JM (2021) An improved stbp for training high-accuracy and low-spike-count spiking neural networks. In: 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 575–580. https://doi.org/10.23919/DATE51398.2021.9474151
38. Wu Y, Deng L, Li G et al (2018) Spatio-temporal backpropagation for training high-performance spiking neural networks. Front Neurosci 12:331. https://doi.org/10.3389/fnins.2018.00331
39. Wu Y, Deng L, Li G, et al (2019) Direct training for spiking neural networks: Faster, larger, better. In: Proceedings of the AAAI conference on artificial intelligence, pp 1311–1318. https://doi.org/10.1609/aaai.v33i01.33011311
40. Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. https://doi.org/10.48550/arXiv.1708.07747
41. Yang G (2022) Asymptotic tracking with novel integral robust schemes for mismatched uncertain nonlinear systems. Int J Robust Nonlinear Control. https://doi.org/10.1002/rnc.6499
42. Yang G, Yao J (2022) Multilayer neuroadaptive force control of electro-hydraulic load simulators with uncertainty rejection. Appl Soft Comput 130(109):672. https://doi.org/10.1016/j.asoc.2022.109672
43. Yang G, Yao J, Dong Z (2022) Neuroadaptive learning algorithm for constrained nonlinear systems with disturbance rejection. Int J Robust Nonlinear Control 32(10):6127–6147. https://doi.org/10.1002/rnc.6143
44. Zenke F, Ganguli S (2018) Superspike: Supervised learning in multilayer spiking neural networks. Neural Comput 30(6):1514–1541. https://doi.org/10.1162/neco_a_01086
45. Zhang L, Zhou S, Zhi T et al (2019) Tdsnn: from deep neural networks to deep spike neural networks with temporal-coding. Proc AAAI Conf Artif Intell 33(1):1319–1326. https://doi.org/10.1609/aaai.v33i01.33011319
46. Zhang M, Wang J, Wu J et al (2021) Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. IEEE Trans Neural Netw Learn Syst 33(5):1947–1958. https://doi.org/10.1109/TNNLS.2021.3110991
47. Zhang W, Li P (2019) Spike-train level backpropagation for training deep recurrent spiking neural networks. In: Advances in Neural Information Processing Systems, vol 32. Curran Associates, Inc
48. Zhang W, Li P (2020) Temporal spike sequence learning via backpropagation for deep spiking neural networks. In: Advances in Neural Information Processing Systems, vol 33. Curran Associates, Inc
49. Zhao D, Zeng Y, Li Y (2022) Backeisnn: a deep spiking neural network with adaptive self-feedback and balanced excitatory-inhibitory neurons. Neural Netw 154:68–77. https://doi.org/10.1016/j.neunet.2022.06.036