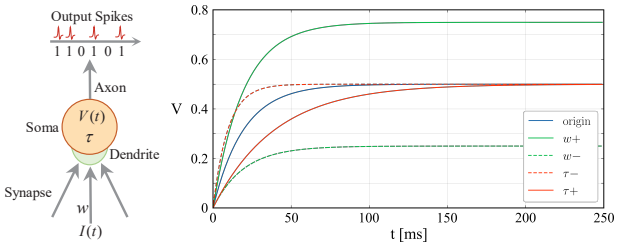
Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks

Wei Fang[1,2], Zhaofei Yu[1,2,3*], Yanqi Chen[1,2], Timothée Masquelier[4], Tiejun Huang[1,2,3], Yonghong Tian[1,2*]
[1]Department of Computer Science and Technology, Peking University, China
[2]Peng Cheng Laboratory, China
[3]Institute for Artificial Intelligence, Peking University, China
[4]Centre de Recherche Cerveau et Cognition (CERCO), UMR5549 CNRS - Univ. Toulouse 3, France
{fwei, yuzf12, chyq}@pku.edu.cn, timothee.masquelier@cnrs.fr, {tjhuang, yhtian}@pku.edu.cn

Abstract

Spiking Neural Networks (SNNs) have attracted enormous research interest due to temporal information processing capability, low power consumption, and high biological plausibility. However, the formulation of efficient and high-performance learning algorithms for SNNs is still challenging. Most existing learning methods learn weights only, and require manual tuning of the membrane-related parameters that determine the dynamics of a single spiking neuron. These parameters are typically chosen to be the same for all neurons, which limits the diversity of neurons and thus the expressiveness of the resulting SNNs. In this paper, we take inspiration from the observation that membrane-related parameters are different across brain regions, and propose a training algorithm that is capable of learning not only the synaptic weights but also the membrane time constants of SNNs. We show that incorporating learnable membrane time constants can make the network less sensitive to initial values and can speed up learning. In addition, we reevaluate the pooling methods in SNNs and find that max-pooling will not lead to significant information loss and have the advantage of low computation cost and binary compatibility. We evaluate the proposed method for image classification tasks on both traditional static MNIST, Fashion-MNIST, CIFAR-10 datasets, and neuromorphic N-MNIST, CIFAR10-DVS, DVS128 Gesture datasets. The experiment results show that the proposed method outperforms the state-of-the-art accuracy on nearly all datasets, using fewer time-steps. Our codes are available at https://github.com/fangwei123456/Parametric-Leaky-Integrate-and-Fire-Spiking-Neuron.

*Corresponding author



(a) Spiking neuron (b) The membrane potential of a LIF neuron

Figure 1. (a) A Leaky Integrate-and-Fire (LIF) neuron with membrane potential V, membrane time constant τ, input $I(t)$ and synaptic weight w. (b) The membrane potential V of the LIF neuron when constant input is received. Increasing or decreasing τ will stretch the $v = f(t)$ curve in the t direction while increasing or decreasing w will stretch the $v = f(t)$ curve in the V direction.

1. Introduction

Spiking Neural Networks (SNNs) are viewed as the third generation of neural network models, which are closer to biological neurons in the brain [42]. Together with neuronal and synaptic states, the importance of spike timing is also considered in SNNs. Due to their distinctive properties, such as temporal information processing capability, low power consumption [55], and high biological plausibility [18], SNNs increasingly arouse researchers' great interest in recent years. Nevertheless, it remains challenging to formulate efficient and high-performance learning algorithms for SNNs.

Generally, the learning algorithms for SNNs can be divided into unsupervised learning, supervised learning, reward-based learning, and Artificial Neural Network (ANN) to SNN conversion methodologies. Either way, we find that most existing learning methods only consider learning the synaptic-related parameters like synaptic weights and treat the membrane-related parameters as

hyperparameters. These membrane-related parameters like membrane time constants, which determine the dynamics of a single spiking neuron, are typically chosen to be the same for all neurons. Note, however, there exist different membrane time constants for spiking neurons across brain regions [43, 9, 33], which are proved to be essential for the representation of working memory and formulation of learning [22, 59]. Thus simply ignoring different time constants in SNNs will limit the heterogeneity of neurons and thus the expressiveness of the resulting SNNs.

In this paper, we propose a training algorithm that is capable of learning not only the synaptic weights but also membrane time constants of SNNs. As illustrated in Fig. 1, we find that adjustments of the synaptic weight and the membrane time constants have different effects on neuronal dynamics. We show that incorporating learnable membrane time constants is able to enhance the learning of SNNs.

The main contributions of this paper can be summarized as follows:

1) We propose the backpropagation-based learning algorithm using spiking neurons with learnable membrane parameters, referred to as Parametric Leaky Integrate-and-Fire (PLIF) spiking neurons, which better represent the heterogeneity of neurons and thereby enhancing the expressiveness of the SNNs. We show that the SNNs made of PLIF neurons are more robust to initial values and can learn faster than SNNs made of neurons with a fixed time constant.
2) We reevaluate the pooling methods in SNNs and discredit the previous conclusion that max-pooling results in significant information loss. We find that compared to average-pooling, max-pooling is able to better preserve the asynchronous characteristic of neuron firing, as well as reduce the computation cost. Our experiments show that the performance of max-pooling is comparable to average-pooling.
3) We evaluate our methods on both traditional static MNIST [35], Fashion-MNIST [67], CIFAR-10 [34] datasets widely used in ANNs as benchmarks, and neuromorphic N-MNIST [50], CIFAR10-DVS [39], DVS128 Gesture [1] datasets that focus on verifying the network's temporal information processing capability. The proposed method exceeds state-of-the-art accuracy on nearly all tested datasets, using fewer time-steps.

2. Related Works

Unsupervised learning of SNNs The unsupervised learning methods of SNNs are based on biological plausible local learning rules, like Hebbian learning [24] and Spike-Timing-Dependent Plasticity (STDP) [3]. Existing approaches exploited the self-organization principle [62, 12, 31], and STDP-based expectation-maximization algorithm [49, 19]. However, these methods are only suitable for shallow SNNs, and the performance is far below state-of-the-art ANN results.

Reward-based learning of SNNs Reward-based learning of SNNs mimics the way the human brain learns by taking advantage of the reward or punishment signals induced by dopaminergic, serotonergic, cholinergic, or adrenergic neurons [15, 6, 45]. Despite the methods that arise in reinforcement learning, like policy gradient [58, 30], temporal-difference learning [52, 16] and Q-learning [6], some heuristic phenomenological models based on STDP [17, 73] were proposed recently.

ANN to SNN conversion ANN to SNN conversion (ANN2SNN) converts a trained non-spiking ANN to an SNN by using the firing rate of each spiking neuron to approximate the corresponding ReLU activation of an analog neuron [26, 7, 56]. It can get near lossless inference results as an ANN [57, 11], but there is a trade-off between accuracy and latency. To improve accuracy, longer inference latency is needed [21]. ANN2SNN is restricted to rate-coding, which loses the processing capability in temporal tasks. As far as we know, ANN2SNN only works for static datasets, not neuromorphic datasets.

Supervised learning of SNNs SpikeProp [5] was the first supervised learning method for SNNs based on backpropagation, which used a linear approximation to overcome the non-differentiable threshold-triggered firing mechanism of SNNs. Subsequent works included Tempotron [20], ReSuMe [51], and SPAN [44], but they could only be applied to single-layer SNNs. Recently, the surrogate gradient method was proposed and provided another solution to training multi-layer SNNs [38, 28, 75, 63, 60, 37, 29]. It utilized surrogate derivatives to define the derivative of the threshold-triggered firing mechanism. Thus the SNNs could be optimized with gradient descent algorithms as ANNs. Zenke et al. [74, 46] systematically studied the remarkable robustness of surrogate gradient learning and showed that SNNs optimized by the surrogate gradient methods can achieve competitive performance with ANNs. Compared to ANN2SNN, the surrogate gradient method has no restrictions on simulating time-steps because it is not based on rate-coding [64, 74].

Spiking neurons and layers of deep SNNs Spiking neuron and layer models play an essential role in SNNs. Cheng et al. [8] added the lateral interactions between neighboring neurons and get better accuracy and stronger noise-robustness. Zimmer et al. [76] firstly adopt the learnable time constants in LIF neurons for the speech recognition task. Bellec et al. [2] proposed the adaptive threshold spiking neuron to enhance computing and learning capabilities of SNNs, which was improved by [69] with learnable time constants. Rathi et al. [53] suggested using a

learnable membrane leak and firing threshold to finetune SNNs converted from ANNs. Despite this, no systematic research on the effects of learning membrane time constants to SNNs has been conducted so far, which is exactly the aim of this paper. Wu et al. [64] found that normalization layers are also critical for deep SNNs and proposed Neuron Normalization (NeuNorm) to balance each neuron's firing rate to avoid severe information loss. Ledinauskas, E et al. [36] firstly suggested that using Batch Normalization [27] in deep SNNs for faster convergence.

3. Methods

In this section, we first briefly review the Leaky Integrate-and-Fire model in Sec. 3.1, and analyze the effect of synaptic weight and membrane time constant in Sec. 3.2. The Parametric Leaky Integrate-and-Fire model and the network structure of the SNNs are then introduced in Sec. 3.3 – Sec. 3.5. At last, we describe the spike max-pooling and the learning algorithm of SNNs in Sec. 3.6 and Sec. 3.7.

3.1. Leaky Integrate-and-Fire model

The basic computing unit of an SNN is the spiking neuron. Neuroscientists have built several spiking neuron models for describing the accurate relationships between input and output signals of the biological neuron. The Leaky Integrate-and-Fire (LIF) model [18] is one of the simplest spiking neuron models used in SNNs. The subthreshold dynamics of the LIF neuron is defined as:

$$\tau \frac{\mathrm{d}V(t)}{\mathrm{d}t} = -(V(t) - V_{rest}) + X(t), \tag{1}$$

where $V(t)$ represents the membrane potential of the neuron at time t, $X(t)$ represents the input to neuron at time t, τ is the membrane time constant, and V_{rest} is the resting potential. When the membrane potential $V(t)$ exceeds a certain threshold V_{th} at time t^f, the neuron will elicit a spike and then the membrane potential $V(t)$ goes back to a reset value $V_{reset} < V_{th}$. The LIF neuron achieves a balance between computing cost and biological plausibility. We set $V_{rest} = V_{reset}$ in this paper, and will not make a distinction between them in the rest of this paper.

3.2. Function comparison of synaptic weight and membrane time constant

In most of the previous learning algorithms for SNNs made of LIF neurons, the membrane time constant τ is regarded as a hyper-parameter and chosen to be the same for all neurons before learning. The learning of SNNs is only to optimize the synaptic weights. However, it cannot be ignored that the behavior of a spiking neuron for given inputs depends not only on the weights of connected synapses but also on the neuron's inherent dynamics controlled by the membrane time constant τ.

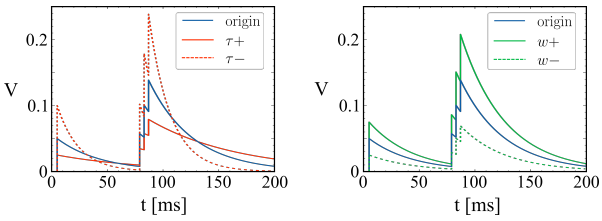


Figure 2. The membrane potential V of a LIF neuron when instant spikes at $t = 5, 80, 85, 90$ are received.

In order to compare the effects of synaptic weight and membrane time constant to the neuronal dynamics, we consider a simple case where the LIF neuron z_i receives weighted input $X(t) = wI(t)$ from a presynaptic neuron z_j (Fig. 1(a)). The rest potential V_{rest} is set to 0. When the input is constant, namely, $I(t) = I$, the membrane potential of the LIF neuron z_i changes over time is shown in Fig. 1(b) (blue curve), which is computed according to Eq. (1). Increasing or decreasing w, as shown by the $w+$ and $w-$ curves, will stretch the $v = f(t)$ curve in the V direction. On the contrary, increasing or decreasing τ will stretch the $v = f(t)$ curve in the t direction, and will not change the steady-state voltage of the neuron z_i as $V(+\infty) = wI$. Fig. 2 illustrates the response of the neuron z_i to instant input spikes at time $t = \{5, 80, 85, 90\}$ms, namely, $X(t) = w(\delta(t-5)+\delta(t-80)+\delta(t-85)+\delta(t-90))$ [1]. The neuron's response to instant input spike at $t = 5$ indicates that a smaller τ (the $\tau-$ curve) leads to faster charge to the steady-state voltage and faster decay to the resting value, making the LIF neuron more sensitive to an instant spike. This sensitivity helps the neuron to capture instant variety in the input. In contrast, a smaller w (the $w-$ curve) leads to a slower charge to the steady-state voltage without affecting decaying speed. When there are three successive input spikes, the membrane potential of the neuron with a smaller τ (the $\tau-$ curve) will reach a higher value at a faster rate, which makes it easier to fire.

To some extent, the effect of decreasing τ is similar to that of increasing w. Nevertheless, adjusting both τ and w can bring some superior additional benefits. As mentioned above, changing both τ and w can stretch the $v = f(t)$ curve, namely the neuron's response to a given input, in both t direction and V direction, which endows the neuron better fitting ability.

3.3. Parametric Leaky Integrate-and-Fire model

We propose the Parametric Leaky Integrate-and-Fire (PLIF) spiking neuron model to learn both the synaptic weights and the membrane time constants of SNNs. The dynamics of the PLIF neuron can be described by Eq. (1).

The SNNs with PLIF neurons follow the three rules:

[1] $\delta(t)$ represents Dirac delta function. If $x \neq 0$, then $\delta(t) = 0$. $\int_{-\infty}^{\infty} \delta(t)\, dt = 1$.

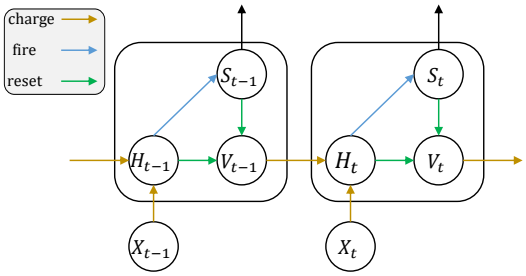


Figure 3. The general discrete spiking neuron model.

(1). The membrane time constant τ is optimized automatically during training, rather than being set as a hyper-parameter manually before training.

(2). The membrane time constant τ is shared within the neurons in the same layer in SNNs, which is biologically plausible as the neighboring neurons have similar properties.

(3). The membrane time constant τ of neurons in different layers are distinct, making diverse phase-frequency responsiveness of neurons.

In fact, the proposed rules are able to increase the heterogeneity of neurons and the expressiveness of the resulting SNNs while effectively controlling computation costs.

For numerical simulations of PLIF neurons in SNNs, we need to consider a version of the parameters dynamics that is discrete in time. Specifically, by including the threshold-triggered firing mechanism and the reset of the membrane potential after firing, we can describe the dynamics of all kinds of spiking neurons with the following equations:

$$H_t = f(V_{t-1}, X_t), \tag{2}$$

$$S_t = \Theta(H_t - V_{th}), \tag{3}$$

$$V_t = H_t\,(1 - S_t) + V_{reset}\,S_t. \tag{4}$$

To avoid confusion, we use H_t and V_t to represent the membrane potential after neuronal dynamics and after the trigger of a spike at time-step t, respectively. X_t denotes the external input, and V_{th} denotes the firing threshold. S_t denotes the output spike at time t, which equals 1 if there is a spike and 0 otherwise. Eq. (3) describes the spike generative process, where $\Theta(x)$ is the Heaviside step function and is defined by $\Theta(x) = 1$ for $x \geq 0$ and $\Theta(x) = 0$ for $x < 0$. Eq. (4) illustrates that the membrane potential returns to V_{reset} after eliciting a spike, which is called *hard reset* and widely used in deep SNNs [36].

As shown in Fig. 3, Eqs. (2) - (4) build a general model to describe the discrete spiking neuron's action: charging, firing, and resetting. Specifically, Eq. (2) describes the neuronal dynamics, and different spiking neuron models have different functions $f(\cdot)$. For example, the function $f(\cdot)$ for the LIF neuron and PLIF neuron is

$$H_t = V_{t-1} + \frac{1}{\tau}(-(V_{t-1} - V_{reset}) + X_t). \tag{5}$$

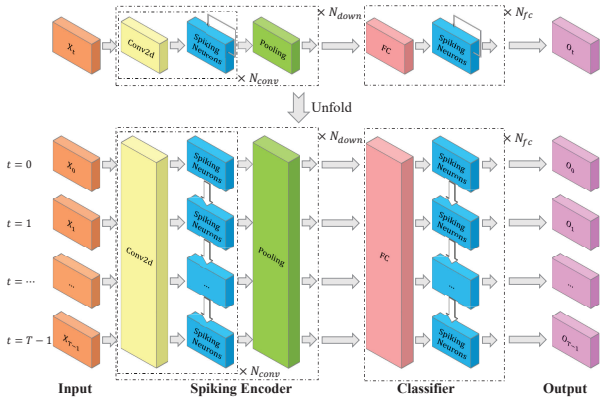


Figure 4. The general formulation of our networks and its unfolded formulation. $\times N_{conv}$ indicates there are N_{conv} {*Conv2d-Spiking Neurons*} connected sequentially. $\times N_{down}$ and $\times N_{fc}$ have the same meaning. Note that the network's parameters are shared at all time-steps.

For PLIF neurons, directly optimizing the membrane time constant τ in Eq. (5) may induce numerical instability as τ is in the denominator. Besides, Eq. (5), as the discrete version of Eq. (1), is a valid approximation only when the time-step $\mathrm{d}t$ is smaller than τ, that is, $\tau > 1$, which is ignored by [53, 69]. To avoid the above problems, we reformulate Eq. (5) to the following equation with a trainable parameter a:

$$H_t = V_{t-1} + k(a)(-(V_{t-1} - V_{reset}) + X_t). \tag{6}$$

Here $k(a)$ denotes the clamp function and $k(a) \in (0, 1)$, which ensures that $\tau = \frac{1}{k(a)} \in (1, +\infty)$. In our experiments, $k(a)$ is the sigmoid activation function, that is, $k(a) = \frac{1}{1+\exp(-a)}$.

3.4. RNN-like Expression of LIF and PLIF

The LIF and PLIF neurons have a similar function as recurrent neural networks. Specifically, when $V_{reset} = 0$, the neuronal dynamics of the LIF neuron and PLIF neuron (Eq. (5)) can be written as:

$$H_t = \left(1 - \frac{1}{\tau}\right) V_{t-1} + \frac{1}{\tau} X_t, \tag{7}$$

where the integration progress $\frac{1}{\tau}X_t$ makes the LIF and PLIF neurons able to remember current input information, while the leakage progress $(1 - \frac{1}{\tau})V_{t-1}$ can be seen as forgetting some information from the past. Eq. (7) shows that the balance between remembrance and forgetting is controlled by the membrane time constant τ, which plays an analogous role as the gates in Long Short-Term Memory (LSTM) networks [25].

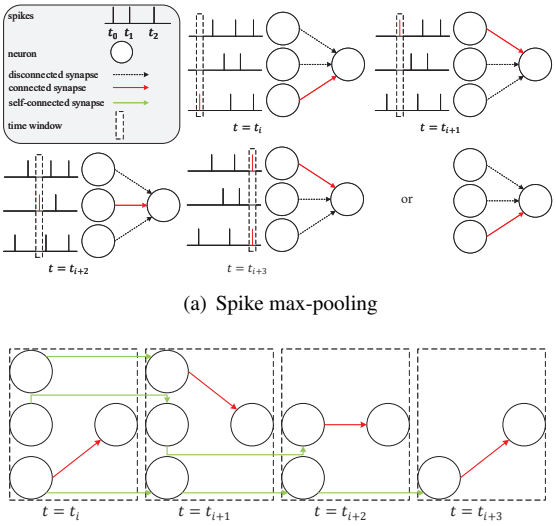
3.5. Network Formulation

We propose a general formulation to build SNNs in this paper, which is illustrated in Fig. 4. The SNN includes a spiking encoder network and a classifier network. The spiking encoder network consists of N_{down} down-sample modules, each of which contains N_{conv} repeated $\{$*Conv2d-Spiking Neurons*$\}$ and a pooling layer. The spiking encoder can extract features from inputs and convert them into the firing spikes at different time-steps. The classifier network consists of N_{fc} repeated $\{$*FC-Spiking Neurons*$\}$. Here *Conv2d* denotes the 2D convolutional layer and *FC* denotes the fully connected layer. Many previous works [12, 37, 60, 75, 8, 21] used a Poisson encoder to convert images to spikes as input, while [56] suggested that this encoding introduces variability into the firing of the network and impairs its performance. Similar to [56, 64, 53], the input is directly fed to our network without being firstly converted to spikes. In this situation, the image-spike encoding is done by the first $\{$*Conv2d-Spiking Neurons*$\}$ module, which can be seen as a learnable encoder. Note that synaptic connections, including convolutional layers and fully connected layers, are stateless, while the spiking neuron layers have self-connections in the temporal domain, as the unfolded network formulation shown in Fig. 4. All parameters are shared at all time-steps.

3.6. Spike Max-Pooling

The pooling layer is widely used to reduce the size of feature maps and to extract compact representation in convolutional ANNs, as well as SNNs. Most previous studies [57, 8, 54] preferred to use the average-pooling in SNNs as they found that max-pooling in SNNs leads to significant information loss. We argue that the max-pooling is consistent with the SNNs' temporal information processing ability and can increase SNNs' fitting capability in temporal tasks and reduce the computation cost for the next layer.

Specifically, the max-pooling layers are behind spiking neuron layers in our model (Fig. 4), and the max-pooling operation is carried on spikes. Different from all neurons that transmit information to the next layer equally in the average-pooling window, only the neuron that fires a spike in the max-pooling window can transmit information to the next layer. Therefore, the max-pooling layer introduces the winner-take-all mechanism, allowing the fired neuron to communicate with the next layer and ignoring other neurons in the pooling window. Another attractive property is that the max-pooling layer will regulate connections dynamically (Fig. 5). The spiking neuron's membrane potential V_t will return to V_{reset} after firing a spike. It is hard for a spiking neuron to fire again as recharging needs time. However, if the neurons in the max-pooling window fire asynchronously, they will be connected to the postsynaptic neuron in turn, which makes the postsynaptic neuron resembles to connect a continuously firing presynaptic neuron and easier to fire. The winner-take-all mechanism in the spatial domain and time-variant topology in the temporal domain achieved by max-pooling can increase SNNs' fitting capability in temporal tasks, such as classifying the CIFAR10-DVS dataset. It is worth noting that the outputs of the max-pooling layer are still binary, while the outputs of the average-pooling layer are float. The matrix multiplication and element-wise multiplication operation on spikes can be accelerated by replacing *multiplication* $*$ with *logical AND* $\&$, which is also the advantage of SNNs compared with ANNs.



(a) Spike max-pooling

(b) Unfolded computation graph

Figure 5. Spike max-pooling regulates connections dynamically. (a) An example of three presynaptic neurons and one postsynaptic neuron with spike max-pooling. At every time-step, only the neuron that fires a spike can connect to the postsynaptic neuron. When more than one neuron fire at the same time-step, the neuron that can connect to the postsynaptic neuron is randomly selected. (b) The unfolded computation graph of (a).

3.7. Training Framework

Here we combine the neuron model (Fig. 3) and network formulation (Fig. 4) to drive the backpropagation training algorithm for SNNs. Denote the simulating time-steps as T and classes number as C, the output $\boldsymbol{O} = [o_{t,i}]$ is a $C \times T$ tensor. For a given input with label l, we encourage the neuron that represents class l to have the highest excitatory level while other neurons should remain silent. So the target output is defined by $\boldsymbol{Y} = [y_{t,i}]$ with $y_{t,i} = 1$ for $i = l$ and $y_{t,i} = 0$ for $i \neq l$. The loss function is defined by the mean squared error (MSE) $L = MSE(\boldsymbol{O}, \boldsymbol{Y}) = \frac{1}{T}\sum_{t=0}^{T-1} L_t = \frac{1}{T}\sum_{t=0}^{T-1}\frac{1}{C}\sum_{i=0}^{C-1}(o_{t,i} - y_{t,i})^2$. And the predicted label l_p is regarded as the index of the neuron with the maximum firing rate $l_p = \arg\max_i \frac{1}{T}\sum_{t=0}^{T-1} o_{t,i}$.

Here we suppose that a^i represents the learnable parameter of the PLIF neurons in the i-th layer in the network. At time-step t, the vectors $\boldsymbol{H}_t^i$ and $\boldsymbol{V}_t^i$ represent the membrane potential after neuronal dynamics and after reset, the vector $\boldsymbol{V}_{th}^i$ and $\boldsymbol{V}_{reset}^i$ represents the threshold and reset potential, respectively. The weighted inputs from the previous layer are $\boldsymbol{X}_t^i = \boldsymbol{W}^{i-1}\boldsymbol{I}_t^i$. $\boldsymbol{S}_t^i = [s_{t,j}^i]$ denotes the output spike at time-step t, where $s_{t,j}^i = 1$ if the j-th neuron fires a spike, else $s_{t,j}^i = 0$. The gradients backward from the next layer are $\frac{\partial L_t}{\partial \boldsymbol{S}_t^i}$. According to Fig. 3 and Fig. 4, we can calculate the gradients recursively:

$$\frac{\partial L}{\partial \boldsymbol{H}_t^i} = \frac{\partial L}{\partial \boldsymbol{H}_{t+1}^i}\frac{\partial \boldsymbol{H}_{t+1}^i}{\partial \boldsymbol{H}_t^i} + \frac{\partial L_t}{\partial \boldsymbol{H}_t^i} \tag{8}$$

$$\frac{\partial \boldsymbol{H}_{t+1}^i}{\partial \boldsymbol{H}_t^i} = \frac{\partial \boldsymbol{H}_{t+1}^i}{\partial \boldsymbol{V}_t^i}\frac{\partial \boldsymbol{V}_t^i}{\partial \boldsymbol{H}_t^i} \tag{9}$$

$$\frac{\partial L_t}{\partial \boldsymbol{H}_t^i} = \frac{\partial L_t}{\partial \boldsymbol{S}_t^i}\frac{\partial \boldsymbol{S}_t^i}{\partial \boldsymbol{H}_t^i} \tag{10}$$

According to Eq. (6), Eq. (3), and Eq. (4) we can get

$$\frac{\partial \boldsymbol{H}_{t+1}^i}{\partial \boldsymbol{V}_t^i} = 1 - k(a^i) \tag{11}$$

$$\frac{\partial \boldsymbol{V}_t^i}{\partial \boldsymbol{H}_t^i} = 1 - \boldsymbol{S}_t + (\boldsymbol{V}_{reset}^i - \boldsymbol{H}_t^i)\frac{\partial \boldsymbol{S}_t^i}{\partial \boldsymbol{H}_t^i} \tag{12}$$

$$\frac{\partial \boldsymbol{S}_t^i}{\partial \boldsymbol{H}_t^i} = \Theta'(\boldsymbol{H}_t^i - \boldsymbol{V}_{th}^i) \tag{13}$$

$$\frac{\partial \boldsymbol{H}_t^i}{\partial \boldsymbol{X}_t^i} = k(a^i) \tag{14}$$

$$\begin{aligned}\frac{\partial \boldsymbol{H}_t^i}{\partial a^i} = &(-(\boldsymbol{V}_{t-1}^i - \boldsymbol{V}_{reset}^i) + \boldsymbol{X}_t^i)k'(a^i) \\ &+ \frac{\partial \boldsymbol{H}_t^i}{\partial \boldsymbol{V}_{t-1}^i}\frac{\partial \boldsymbol{V}_{t-1}^i}{\partial \boldsymbol{H}_{t-1}^i}\frac{\partial \boldsymbol{H}_{t-1}^i}{\partial a^i}\end{aligned} \tag{15}$$

Finally, we can get the gradients of the learnable parameters:

$$\frac{\partial L}{\partial a^i} = \sum_{t=0}^{T-1}\frac{\partial L}{\partial \boldsymbol{H}_t^i}\frac{\partial \boldsymbol{H}_t^i}{\partial a^i} \tag{16}$$

$$\frac{\partial L}{\partial \boldsymbol{W}^{i-1}} = \sum_{t=0}^{T-1}\frac{\partial L}{\partial \boldsymbol{H}_t^i}\frac{\partial \boldsymbol{H}_t^i}{\partial \boldsymbol{X}_t^i}\boldsymbol{I}_t^i \tag{17}$$

Note that $\frac{\partial *}{\partial \boldsymbol{S}_t^i} = 0$ when $t \geq T$, $\boldsymbol{V}_{-1}^i = \boldsymbol{V}_{reset}^i$. We use derivative of the surrogate function $\sigma(x)$ to define the derivative of spiking function $\Theta(x)$ (see supplementary). $k(x)$ is the clamp function.

4. Experiments

We evaluate the performance of SNNs with PLIF neurons and spike max-pooling for classification tasks on both traditional static MNIST, Fashion-MNIST, CIFAR-10 datasets, and neuromorphic N-MNIST, CIFAR10-DVS, and DVS128 Gesture datasets. More details of the training can be found in the supplementary.

Dataset	N_{conv}	N_{down}	N_{fc}
*MNIST	1	2	2
CIFAR-10	3	2	2
CIFAR10-DVS	1	4	2
DVS128 Gesture	1	5	2

Table 1. Network structures for different datasets. N_{conv}, N_{down} and N_{fc} are defined in Fig. 4. **MNIST* denotes MNIST, Fashion-MNIST and N-MNIST datasets.

4.1. Network Structure

The network structures of SNNs for different datasets are shown in Tab. 1. We set $kernel\ size = 3$, $stride = 1$ and $padding = 1$ for all *Conv2d* layers. The $out\ channels$ of *Conv2d* layers is 256 for CIFAR-10 dataset and 128 for all other datasets. A batch normalization (*BN*) layer is added after each *Conv2d* layer. As the parameters of a *BN* layer can be absorbed in its front *Conv2d* layer [56], we can remove *BN* in the SNNs for inference. All pooling layers set $kernel\ size = 2$ and $stride = 2$. For all networks, the $out\ features$ of the first FC layer is a quarter of the $in\ features$, and the $out\ features$ of the second FC layer is $M \cdot C$, where C is the classes number and M is the neurons of a population to represent one class. A dropout layer [37] is placed before each FC layer. A voting layer after the output spiking neurons layer is used to boost classifying robustness. The voting layer is implemented by average-pooling with $kernel\ size = M$ and $stride = M$. We set $M = 10$ for all datasets. We use the average-pooling to implement democratic voting, such that the minority is subordinate to the majority. Using max-pooling to vote may result in a dictatorship, as the minority will not be involved in the computation graph (see Fig. 5) and using M neurons to represent one class will degenerate into using one neuron.

4.2. Comparison with the State-of-the-Art

Tab. 2 shows the accuracies of the proposed methods (PLIF neurons with $\tau_0 = 2$, max-pooling) and other comparing methods on both traditional static MNIST, Fashion-MNIST, CIFAR-10 datasets, and neuromorphic N-MNIST, CIFAR10-DVS, DVS128 Gesture datasets. We set the same training hyperparameters for all datasets (see supplementary). As shown in Tab. 2, we achieve the highest accuracies on all datasets except for CIFAR-10. The accuracy on CIFAR-10 is slightly lower than [21], which is based on ANN2SNN conversion. However, they only applied to static images as ANN2SNN is ill-suited to neuromorphic datasets. Different from them, our method is also applicable to neuromorphic datasets and outperforms the spike-based

Model	Method	Accuracy MNIST	Accuracy Fashion-MNIST	Accuracy CIFAR-10	Accuracy N-MNIST	Accuracy CIFAR10-DVS	Accuracy DVS128 Gesture
[26]	ANN2SNN	98.37%	-	82.95%	-	-	-
[56]	ANN2SNN	99.44%	-	88.82%	-	-	-
[57]	ANN2SNN	-	-	91.55%	-	-	-
[21]	ANN2SNN	-	-	**93.63%**	-	-	-
[38]	Spike-based BP	99.31%	-	-	98.74%	-	-
[63]	Spike-based BP	99.42%	-	-	98.78%	50.7%	-
[60]	Spike-based BP	99.36%	-	-	99.2%	-	93.64%
[29]	Spike-based BP	-	-	-	96%	-	**95.54%**
[28]	Spike-based BP	99.49%	-	-	98.84%	-	-
[75]	Spike-based BP	**99.62%**	90.13%	-	-	-	-
[64]	Spike-based BP	-	-	90.53%	**99.53%**	**60.5%**	-
[37]	Spike-based BP	99.59%	-	90.95%	99.09%	-	-
[8]	Spike-based BP	99.5%	**92.07%**	-	99.45%	-	-
[40]	Spike-based BP	-	-	-	96.3%	32.2%	-
[68]	Spike-based BP	-	-	-	-	-	92.01%
[13]	Spike-based BP	99.46%	-	-	99.39%	-	96.09% (10 classes)
[23]	Spike-based BP	-	-	-	98.28%	-	93.40%
[53]	ANN2SNN and Spike-based BP	-	-	92.64%	-	-	-
[61]	HATS	-	-	-	99.1%	52.4%	-
[4]	GCN	-	-	-	99.0%	54.0%	-
Ours	Spike-based BP	99.72%	94.38%	93.50%	99.61%	74.80%	97.57%

Table 2. Performance comparison between the proposed method and the state-of-the-art methods on different datasets. The highest accuracies of previous works are in bold.

Dataset	SOTA	SOTA's T	ours T
MNIST	[75]	400	8
Fashion-MNIST	[8]	20	8
CIFAR-10	[21]	2048	8
N-MNIST	[64]	59-64	10
CIFAR10-DVS	[64]	230-292	20
DVS128 Gesture	[29]	500(training) 1800(testing)	20

Table 3. The time-steps of previous SOTA works and ours on each dataset.

BP SOTA accuracy.

Tab. 3 compares the number of time-steps of our method and the previous works that achieve the best performance on each dataset. It can be found that the proposed method takes fewer time-steps than all the other methods. For example, our method uses up to $256\times$ fewer inference time-steps compared to ANN2SNN conversion [21]. Thus our method can not only decrease the memory consumption and the training time but also increase inference speed greatly.

4.3. Ablation Study

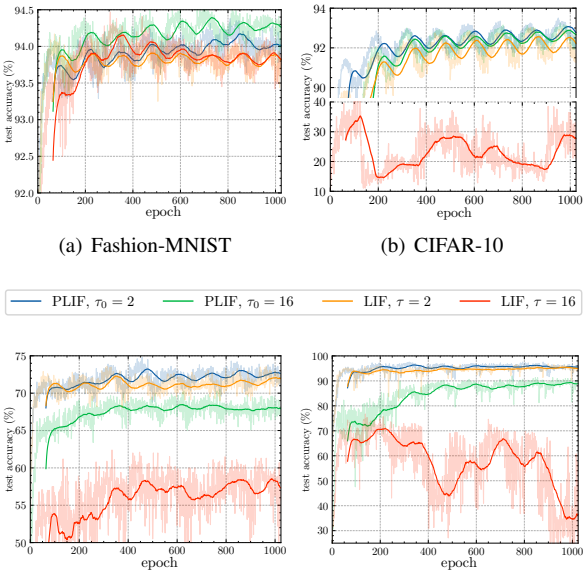
We conduct extensive ablation studies to evaluate PLIF neurons and max-pooling on four challenging datasets. We

Neuron	Fashion-MNIST	CIFAR-10	CIFAR10-DVS	DVS128 Gesture
PLIF($\tau_0 = 2$)	**94.38%**	**93.50%**	**74.80%**	**97.57%**
LIF($\tau = 2$)	94.17%	93.03%	73.60%	96.88%
PLIF($\tau_0 = 16$)	**94.65%**	**93.23%**	**70.50%**	**92.01%**
LIF($\tau = 16$)	94.47%	47.50%	62.40%	76.74%

Table 4. Accuracy of using PLIF/LIF.

first study the effect of PLIF neurons. In this experiment, we train the same SNNs with PLIF neurons and LIF neurons respectively, and compare the test accuracy. As shown in Tab. 4, if the initial membrane time constant τ_0 of PLIF neurons is set equal to the membrane time constant τ of LIF neurons, the test accuracy of the SNNs with PLIF neurons is always higher than that with LIF neurons. This is due to the membrane time constants of PLIF neurons in different layers can be different after learning, which better represents the heterogeneity of neurons. Fig. 6 illustrates the test accuracy of PLIF vs. LIF neurons during training. As can be seen, the accuracy and convergence speed of the SNNs with LIF neurons decrease seriously if the initial value of the membrane time constant is not reasonable (red curve). In contrast, the PLIF neurons can learn the appropriate membrane time constants and achieve better performance (green curve).

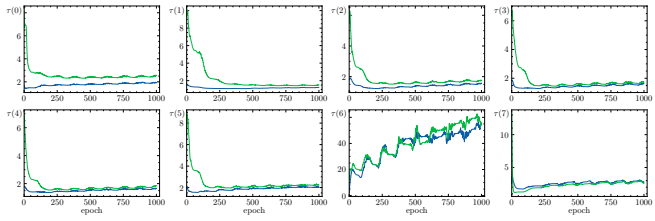
To analyze the influence of initial values in PLIF neu-



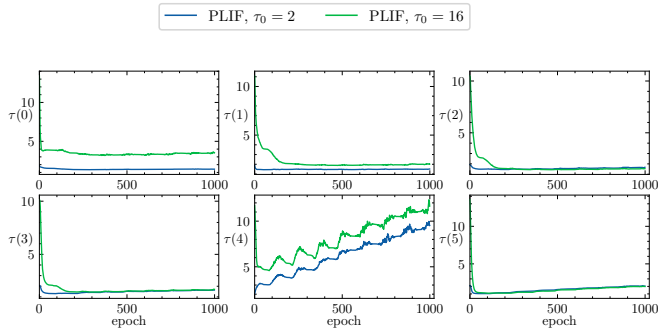
(a) Fashion-MNIST (b) CIFAR-10

(c) CIFAR10-DVS (d) DVS128 Gesture

Figure 6. The test accuracy of PLIF vs. LIF neurons on different datasets during training. The shaded curves indicate the origin data. The solid curves are 64-epoch moving averages.



(a) The change of $\tau(i)$ during training on CIFAR-10.



(b) The change of $\tau(i)$ during training on CIFAR10-DVS.

Figure 7. The change of membrane time constants in different layers during training with different initial values. $\tau(i)$ represents the membrane time constant τ of the i-th PLIF neurons layer.

rons, we show how the membrane time constants of the neurons in each layer change during learning with respect to different initial values. As shown in Fig. 7, the membrane time constants with different initial values in each layer tend to gather during training, which indicates that the

Pooling	Fashion-MNIST	CIFAR-10	CIFAR10-DVS	DVS128 Gesture
Average	**94.74%**	93.30%	72.70%	97.22%
Max	94.38%	**93.50%**	**74.80%**	**97.57%**

Table 5. Accuracy of using max-pooling/average-pooling.

PLIF neurons are robust to initial values. Note that $\tau(6)$ in Fig. 7(a) and $\tau(4)$ in Fig. 7(b) tend to infinity. This could be explained as follows. The PLIF neurons with the membrane time constants $\tau(4)$ and $\tau(6)$ in two SNNs are behind the first FC layer with weight W_{fc}. We check the training logs and find that the distribution, mean and variance of $\frac{W_{fc}}{\tau}(\tau = \tau(4)$ or $\tau(6))$ converge after dozens of epochs (see supplementary). Refer to the dynamics of PLIF neurons (Eq. (5)) with $X_t = W_{fc}I_t$ and $\frac{1}{\tau} \to 0$, we can find $H_t \to V_{t-1} + \frac{W_{fc}}{\tau}I_t$. It means that the PLIF neurons after the first FC layer are learning to become the Non-Leaky-Integrate-and-Fire neurons.

We further study the effect of max-pooling. Tab. 5 compares the accuracy of the proposed SNNs with max-pooling/average-pooling on four challenging datasets. The performance of max-pooling is similar to that of average-pooling, which indicates that the previous conclusion that max-pooling results in significant information loss in SNNs is not reasonable. Remarkably, the max-pooling gets slightly higher accuracies on CIFAR-10, CIFAR10-DVS, and DVS128 Gesture datasets, showing its better fitting capability in complex tasks.

5. Conclusion

In this work, we proposed the Parametric Leaky Integrate-and-Fire (PLIF) neuron to incorporate the learnable membrane time parameter into SNNs. We show that the SNNs with the PLIF neurons outperform state-of-the-art comparing methods on both static and neuromorphic datasets. Besides, we show that the SNNs made of PLIF neurons are more robust to initial values and can learn faster than SNNs consist of LIF neurons. We also reevaluate the performance of max-pooling and average-pooling in SNNs and find the previous works underestimate the performance of max-pooling. We recommend using max-pooling in SNNs for its lower computation cost, higher temporal fitting capability, and the characteristic to receive spikes and output spikes rather than floating values as average-pooling.

6. Acknowledgment

This work is supported by grants from the National Natural Science Foundation of China under contracts No.62027804, No.61825101, and No.62088102.

A. Supplementary Materials

B. Reproducibility

All experiments are implemented by SpikingJelly [14]. All of the source codes, training logs are available on GitHub. To maximize reproducibility, we use identical seeds in all codes.

C. Network Structure Details

Tab. 6 illustrates the details of the network structures for different datasets. *c128k3s1* represents the convolutional layer with $output\ channels = 128$, $kernel\ size = 3$ and $stride = 1$. *BN* is the batch normalization. *MPk2s2* is the max-pooling layer with $kernel\ size = 2$ and $stride = 2$. *PLIF* is the PLIF spiking neurons layer. *DP* represents the dropout layer [37]. *FC2048* represents the fully connected layer with $output\ features = 2048$. The symbol $\{\}$* indicates the repeated structure. For example, $\{$*c128k3s1-BN-PLIF-MPk2s2*$\}$*2 means that there are two $\{$*c128k3s1-BN-PLIF-MPk2s2*$\}$ modules connected sequentially. The last layer *APk10s10* is the voting layer, which is implemented by an average-pooling layer with $kernel\ size = 10$ and $stride = 10$.

D. Training Algorithm to Fit Target Output

After defining the derivative of the spike generative process, the parameters of SNNs can be trained by gradient descent algorithms as that in ANNs. Classification, which is the task in this paper, as well as other tasks for both ANNs and SNNs, can be seen as optimizing parameters of the network to fit a target output when given a specific input. The gradient descent algorithm for SNNs to fit a target output is derived in the main text (Eq.(16) and Eq.(17)), and is as follows:

Dataset	Network Structure
*MNIST	{c128k3s1-BN-PLIF-MPk2s2}*2-DP-FC2048-PLIF-DP-FC100-PLIF-APk10s10
CIFAR-10	{{c256k3s1-BN-PLIF}*3-MPk2s2}*2-DP-FC2048-PLIF-DP-FC100-PLIF-APk10s10
CIFAR10-DVS	{c128k3s1-BN-PLIF-MPk2s2}*4-DP-FC512-PLIF-DP-FC100-PLIF-APk10s10
DVS128 Gesture	{c128k3s1-BN-PLIF-MPk2s2}*5-DP-FC512-PLIF-DP-FC110-PLIF-APk10s10

Table 6. Detailed network structures for different datasets. **MNIST* represents MNIST, Fashion-MNIST, and N-MNIST datasets.

Algorithm 1 Gradient Descent Algorithm for SNNs to Fit Target Output

Require: learning rate ϵ, network's parameter $\boldsymbol{\theta}$, total simulating time-steps T, input $\boldsymbol{X} = \{\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_{T-1}\}$, target output $\boldsymbol{Y} = \{\boldsymbol{Y}_0, \boldsymbol{Y}_1, ..., \boldsymbol{Y}_{T-1}\}$, loss function $L = \mathcal{L}(\boldsymbol{O}, \boldsymbol{Y})$
initialize $\boldsymbol{\theta}$
create an empty list $\boldsymbol{S} = \{\}$
for $t \leftarrow 0, 1, ...T-1$
  input $\boldsymbol{X}_t$ to network, get output spikes $\boldsymbol{S}_t$
  append $\boldsymbol{S}_t$ to $\boldsymbol{S} = \{\boldsymbol{S}_0, \boldsymbol{S}_1, ..., \boldsymbol{S}_{t-1}\}$
calculate loss $L = \mathcal{L}(\boldsymbol{Y}, \boldsymbol{O})$
update parameter $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \cdot \nabla_{\boldsymbol{\theta}} L$

Here the loss function $L = \mathcal{L}(\boldsymbol{O}, \boldsymbol{Y})$ is a distance measurement between $\boldsymbol{Y}$ and $\boldsymbol{S}$, e.g., the mean squared error (MSE) in the main text.

E. Introduction of the Datasets

MNIST The MNIST dataset of handwritten digits comprises 28×28 gray-scale images which are labeled from 0 to 9. The MNIST dataset includes 60,000 training images and 10,000 test images.

Fashion-MNIST Similar to the MNIST dataset, the Fashion-MNIST dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example in the Fashion-MNIST dataset is a 28×28 gray-scale image with a label from 0 to 9.

CIFAR-10 The CIFAR-10 dataset consists of 60,000 natural images in 10 classes, with 6,000 images per class. The number of the training images is 50,000, and that of the test images is 10,000.

N-MNIST The Neuromorphic-MNIST (N-MNIST) dataset is a spiking version of the MNIST dataset recorded by the neuromorphic sensor. It was converted from MNIST by mounting the ATIS sensor on a motorized pan-tilt unit and moving the sensor while recording MNIST examples on an LCD monitor. It consists of 60,000 training examples and 10,000 test examples.

CIFAR10-DVS The CIFAR10-DVS dataset is the neuromorphic version of the CIFAR-10 dataset. It is composed of 10,000 examples in 10 classes, with 1000 examples in each class. As the CIFAR10-DVS dataset does not separate data into training and testing sets, in each class, we choose the first 9000 samples for training and the rest 1000 samples for testing, which is similar to [64].

DVS128 Gesture The DVS128 Gesture dataset is recorded by a DVS128 camera, which contains 11 kinds of hand gestures from 29 subjects under 3 kinds of illumination conditions.

F. Preprocessing

Static Datasets. We apply data normalization on all static datasets to ensure that input images have zero mean and unit variance. Besides, random horizontal flipping and cropping on MNIST and CIFAR-10 are conducted to avoid overfitting. We do not use these augmentations on Fashion-MNIST as images in this dataset are tidy.

Neuromorphic Datasets. The data in neuromorphic datasets usually take the form of address event representation (AER) $E(x_i, y_i, t_i, p_i)$ $(i = 0, 1, ..., N-1)$ to represent the event location in the asynchronous stream, the timestamp, and the polarity. As the number of events is large, e.g. more than one million in CIFAR10-DVS, we split the events into T slices with nearly the same number of events in each slice and integrate events to frames. The new representation $F(j, p, x, y)$ ($0 \leq j \leq T-1$) is the summation of event data in the j-th slice:

$$F(j,p,x,y) = \sum_{i=j_l}^{j_r-1} \mathcal{I}_{p,x,y}(p_i, x_i, y_i), \tag{18}$$

where $\mathcal{I}_{p,x,y}(p_i, x_i, y_i)$ is an indicator function and it equals 1 only when $(p, x, y) = (p_i, x_i, y_i)$. j_l and j_r are the minimal and the maximal timestamp indexes in the j-th slice. $j_l = \lfloor \frac{N}{T} \rfloor \cdot j$, $j_r = \lfloor \frac{N}{T} \rfloor \cdot (j+1)$ if $j < T-1$ and N if $j = T-1$. Here $\lfloor \cdot \rfloor$ is the floor operation. Note that T is also the number of time-steps in our experiments.

Similar event-to-frame integrating methods for preprocessing neuromorphic datasets are widely used in both ANNs [23, 48, 47] and SNNs [64, 65, 8, 68, 23, 29, 38]. T of [64] in Tab. 3 of the main text for N-MNIST and CIFAR10-DVS are calculated manually according to their paper. Specifically, they illustrate that the time resolution is reduced by accumulating the spike train within every 5 ms and the time range (us) of N-MNIST and CIFAR10-DVS are [290901, 315348] and [1149758, 1459301], respectively.

G. Hyper-Parameters

We use the Adam [32] optimizer with the learning rate 0.001 and the cosine annealing learning rate schedule [41] with $T_{schedule} = 64$. The *batch size* is set to 16 to reduce memory consumption. The drop probability p for dropout layers is 0.5. The clamp function for PLIF neurons is $k(a) = \frac{1}{1+e^{-a}}$ and the surrogate gradient function is $\sigma(x) = \frac{1}{\pi}\arctan(\pi x) + \frac{1}{2}$, thus $\sigma'(x) = \frac{1}{1+(\pi x)^2}$. We set $V_{reset} = 0$ and $V_{th} = 1$ for all neurons. We notice that some previous works, e.g., [63], [64], fine tuned V_{th} for different tasks, which is unnecessary. To be specific, as $\Theta(V - V_{th}) = \Theta(V_{th}(\frac{V}{V_{th}} - 1)) = \Theta(\frac{V}{V_{th}} - 1)$ and V is directed influenced by trainable weights, setting $V_{th} = 1$ implements an implicit normalization for weights, which can mitigate the exploding and vanishing gradient problem. As discovered by Zenke and Vogels [74], ignoring the neuronal reset when computing gradients by detaching them from the computational graph can improve performance, we also detach S_t in the neuronal reset.

Dataset	Without Validation	15% Validation
MNIST	99.72%	99.63%
Fashion-MNIST	94.38%	93.85%
CIFAR-10	93.50%	92.58%
N-MNIST	99.61%	99.57%
CIFAR10-DVS	74.80%	69.00%
DVS128 Gesture	97.57%	96.53%

Table 7. Accuracy of the proposed method with/without the validation set on different datasets.

H. Accuracy with a Validation Set

The performance comparison in Tab. 2 of the main text is obtained by training on the training set, testing on the test set alternately, and recording the maximum test accuracy. Both this paper and the state-of-the-art methods use this way to report performance. However, this kind of accuracy is overestimated. Here we also report the accuracy with validation, which is obtained by splitting the origin training set into a new training set and validation set, training on the new training set, testing on the validation set alternately, and recording the test accuracy on the test set only once with the model that achieved the maximum validation accuracy. We utilize 85% samples of each class in the origin training set as the new training set and set the rest 15% as the validation set. The accuracy with and without the validation set of proposed methods is shown at Tab. 7. The experiment results in Tab. 2 of the main text and Tab. 7 show that the proposed method outperforms the state-of-the-art accuracy on nearly all datasets.

I. Distribution of the First $\frac{W_{fc}}{\tau}$

In Sec. 4.3 of the main text, we find that PLIF neurons after the first FC layer are learning to become the Non-Leaky-Integrate-and-Fire neurons as $\frac{1}{\tau} \to 0$ and $\frac{W_{fc}}{\tau}$ converges. To illustrate the convergence, we show the distribution of $\frac{W_{fc}}{\tau}$ during training the SNN on CIFAR-10DVS in Fig. 8. The distribution of $\frac{W_{fc}}{\tau}$ on other datasets converges in the same way.

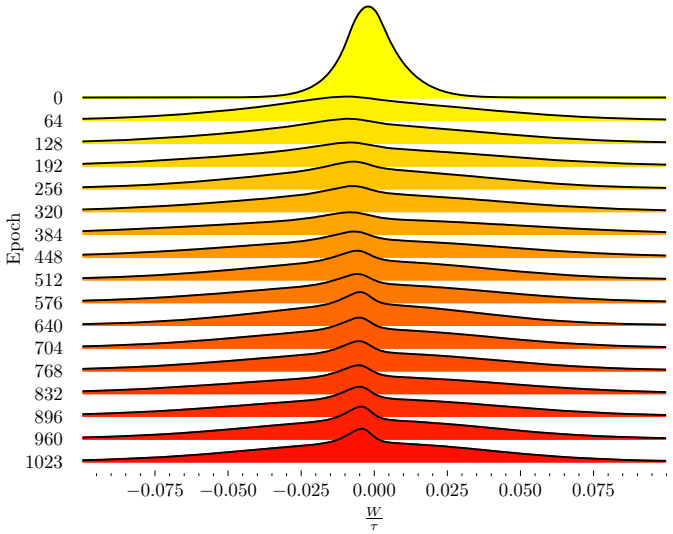


Figure 8. The distribution of $\frac{W_{fc}}{\tau}$ during training the SNN on CIFAR10-DVS.

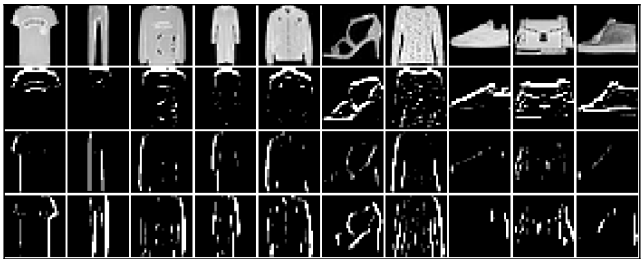


Figure 9. Ten samples from the Fashion-MNIST dataset and the corresponding firing rates $\boldsymbol{F}^2_{T_s=8}$ from channel 45,75 and 76 ($c = 45, 75, 76$) of the first PLIF neurons layer are shown in row 1-4. Each column represents a sample and corresponding firing rates.

J. Visualization of Spiking Encoder

To evaluate the learnable encoder, we give inputs $\boldsymbol{x}_t$ to the trained network and show the output spikes $\boldsymbol{S}^n_t(c)$ and the firing rates $\boldsymbol{F}^n_{T_s}(c) = \frac{1}{T_s}\sum_{t=0}^{T_s-1}\boldsymbol{S}^n_t(c)$ from channel c in the n-th layer, which is similar to [10]. Although the output spikes from deeper spiking neurons layers contain more semantic features, they are harder to read and understand. Thus we only show the spikes from the first spiking neurons layer, that is, $n = 2$.

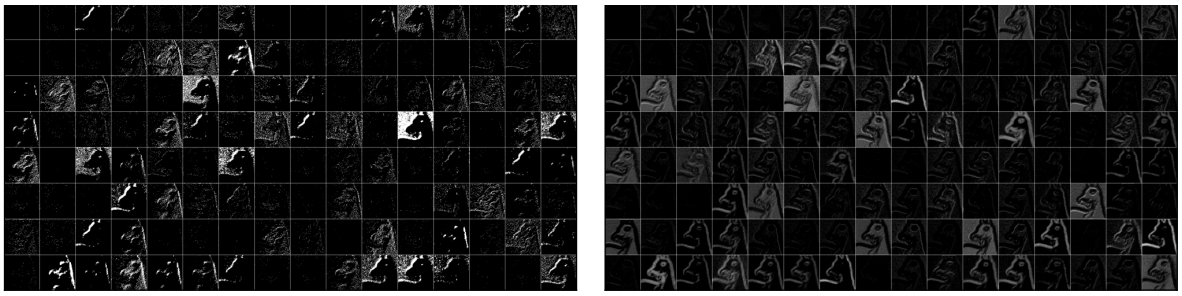
Fig. 9 illustrates 10 input images from static Fashion-MNIST dataset (row 1) and the firing rates $\boldsymbol{F}^2_{T_s=8}$ of three typical channel (45, 75 and 76) of the first PLIF neurons layer (row 2, 3 and 4). One can find that the firing rates from channel 45, 75 and 76 detect upper, left, right edges of the input images. Fig. S10(a) shows a 2-D grid flatten across channels from the 3-D tensor $\boldsymbol{S}^2_{t=0}(c = 0, 1, ..., 127)$ when given an input sample labeled *horse*, which illustrates the features extracted by the spiking encoder at $t = 0$. As the CIFAR10-DVS dataset is converted from the static CIFAR-10 dataset, the firing rates accumulated from spikes can reconstruct the images filtered by the convolutional layer. Fig. S10(b) illustrates the firing rates $\boldsymbol{F}^2_{T_s=19}$ of all 128 channels ($c = 0, 1, ..., 127$), which have clearer texture than binary output spikes in Fig. S10(a). Fig. S11(a) shows the input $\boldsymbol{x}_t$ (row 1) and the corresponding output spikes $\boldsymbol{S}^2_t$ of channel 40 and 103 (row 2 and 3) at $t = 0, 1, ..., 19$, and Fig. S11(b) shows the mean input $\boldsymbol{x}(T_s) = \frac{1}{T_s}\sum_{t=0}^{T_s-1}\boldsymbol{x}_t$ (row 1) and the corresponding firing rates $\boldsymbol{F}^2_{T_s}$ of channel 40 and 103 (row 2 and 3) at $T_s = 0, 1, ..., 19$. One can find that as T_s increases, the texture constructed by firing rates $\boldsymbol{F}^2_{T_s}$ becomes more distinct, which is similar to the use of the Poisson encoder.

Fig. 12 visualizes three input samples $\boldsymbol{x}_t$ and output spikes $\boldsymbol{S}^2_t(c = 59)$ in the DVS128 Gesture dataset. Three samples labeled *random other gestures*, *right hand clockwise*, *drums* at $t = 0, 1, ..., 19$ from the DVS128 Gesture dataset are shown in row 1, 3, 5 of Fig. 12. For comparison, the corresponding output spikes from channel 59 of the PLIF neurons in the first conventional layer are shown in rows 2, 4, 6. One crucial difference is that the output almost only includes the gesture's response spikes, indicating that the spiking neurons implement efficient and accurate filtering on both spatial-variant and temporal-variant input data, reserving the gesture but discarding the player.

K. Relations between different Encoders

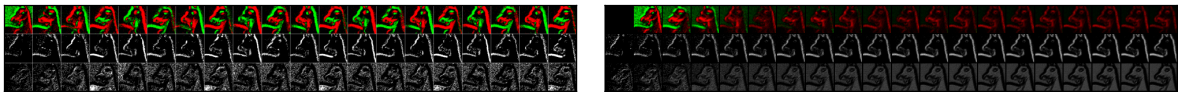
The Poisson encoder is one of the rate encoding methods and widely used in SNNs [12, 37, 60, 75, 8, 21] to encode images into spikes. Given a image pixel $p \in [0, 1]$, the encoded spike S_t at time-step t is fired with the probability p. Thus, the expectation of the number of spikes during the whole time-steps T is $E_{Poisson}(\Sigma_{t=0}^{T-1}S_t) = pT$. In our poposed SNNs, the input is directly fed to the network without being first converted to spikes and the image-spike encoding is done by the first $\{Conv2d\text{-}Spiking\ Neurons\}$ module (BN is omitted), which can be seen as a learnable encoder. Here we denote this encoder as ENC_l. If we set $Conv2d$ non-learnable with $channels = kernel\ size = 1$, the kernel weight as the constant $w > 0$, and $Spiking\ Neurons$ as Non-Leaky-Integrate-and-Fire neurons with threshold potential V_{th} and $V_{reset} = 0$, then the expectation of spikes number of this module is $E(\Sigma_{t=0}^{T-1}S_t) = \lfloor \frac{T}{\lceil \frac{V_{th}}{wp} \rceil} \rfloor$, where $\lceil\rceil$ denotes the ceiling operation. We can find that $E(\Sigma_{t=0}^{T-1}S_t) \approx E_{Poisson}(\Sigma_{t=0}^{T-1}S_t)$ when $V_{th} = w = 1$, which indicates that ENC_l can approximate the function of the Possion encoder in rate encoding.

The latency encoder used in [66, 70, 71, 72] is a representative temporal encoding method. The latency encoder encodes the image pixel p into a spike at time-step t_p. Thus, the information of input is encoded in the precise firing time



(a) $\boldsymbol{S}^2_{t=0}(c = 0, 1, ..., 127)$ (b) $\boldsymbol{F}^2_{T_s=19}(c = 0, 1, ..., 127)$

Figure 10. Given a sample labeled *horse* from CIFAR10-DVS, (a) shows spikes from all 128 channels of the first spiking neurons layer at $t = 0$, and (b) shows firing rates of these neurons at $T_s = 19$.



(a) $\boldsymbol{x}_t$ and $\boldsymbol{S}^2_t(c = 40, 103)$ at $t = 0, 1, ..., 19$ (b) $\boldsymbol{x}(T_s)$ and $\boldsymbol{F}^2_{T_s}(c = 40, 103)$ at $T_s = 0, 1, ..., 19$

Figure 11. Given the sample sample as Fig. 10, the input data and output spikes of channel 40 and 103 at each time-step are showed in (a) at row 1, 2, 3, respectively. The mean input data and firing rates of channel 40 and 103 at each time-step are showed in (b).

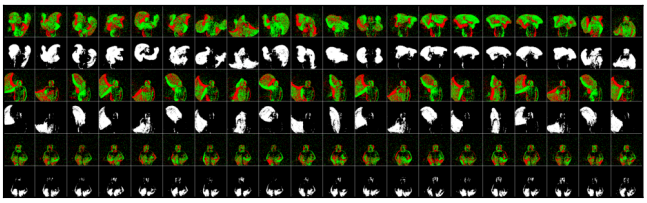


Figure 12. Three samples from the DVS128 Gesture dataset labeled *random other gestures*, *right hand clockwise*, *drums* are shown in row 1, 3, 5. The corresponding output spikes from channel 59 of the first PLIF neurons layer are shown in row 2, 4, 6.

of the spike. t_p is usually inversely proportional to the input intensity p, e.g., $t_p = \lfloor (T_{max} - 1)(1 - p) \rfloor$ and T_{max} is the encoding period. We can also find that the first firing time of ENC_l for the given input p is $\lceil \frac{V_{th}}{wp} \rceil$, which satisfies that the larger input intensity p causes the faster spike. In fact, the latency encoder is an extremely simplified learnable encoder with directly inputted images. In this paper, the proposed learnable encoders have learnable weights and more channels, which is able to encode images into complex spikes pattern with more semantic information, e.g., reserving the gesture but discarding the player of samples from DVS128 Gesture dataset (see Fig. 12).

References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017. 2

[2] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pages 787–797, 2018. 2

[3] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, 1998. 2

[4] Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos. Graph-based object classification for neuromorphic vision sensing. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 491–501, 2019. 7

[5] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002. 2

[6] Matthew Botvinick, Jane X. Wang, Will Dabney, Kevin J. Miller, and Zeb Kurth-Nelson. Deep reinforcement learning and its neuroscientific implications. *Neuron*, 107(4):603–616, 2020. 2

[7] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015. 2

[8] Xiang Cheng, Yunzhe Hao, Jiaming Xu, and Bo Xu. LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition. In *IJCAI*, pages 1519–1525. International Joint Conferences on Artificial Intelligence Organization, 7 2020. 2, 5, 7, 10, 11

[9] Gustavo Deco, Josephine Cruzat, and Morten L Kringelbach. Brain songs framework used for discovering the relevant timescale of the human brain. *Nature Communications*, 10(1):1–13, 2019. 2

[10] Lei Deng, Yujie Wu, Xing Hu, Ling Liang, Yufei Ding, Guoqi Li, Guangshe Zhao, Peng Li, and Yuan Xie. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks*, 121:294–307, 2020. 11

[11] Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2021. 2

[12] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015. 2, 5, 11

[13] Haowen Fang, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. Exploiting Neuron and Synapse Filter Dynamics in Spatial Temporal Learning of Deep Spiking Neural Network. *arXiv preprint arXiv:2003.02944*, 2020. 7

[14] Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. Spikingjelly. https://github.com/fangwei123456/spikingjelly, 2020. 9

[15] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016. 2

[16] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS Computational Biology*, 9(4):e1003024, 2013. 2

[17] Johannes Friedrich, Robert Urbanczik, and Walter Senn. Spatio-temporal credit assignment in neuronal population learning. *PLoS Computational Biology*, 7(6):e1002092, 2011. 2

[18] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. 1, 3

[19] Shangqi Guo, Zhaofei Yu, Fei Deng, Xiaolin Hu, and Feng Chen. Hierarchical bayesian inference and learning in spiking neural networks. *IEEE Transactions on Cybernetics*, 49(1):133–145, 2017. 2

[20] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature Neuroscience*, 9(3):420–428, 2006. 2

[21] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. RMP-SNN: Residual Membrane Potential Neuron for Enabling Deeper High-Accuracy and Low-Latency Spiking Neural Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 13558–13567, 2020. 2, 5, 6, 7, 11

[22] Michael E Hasselmo and Chantal E Stern. Mechanisms underlying working memory for novel information. *Trends in Cognitive Sciences*, 10(11):487–493, 2006. 2

[23] Weihua He, YuJie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. Comparing SNNs and RNNs on Neuromorphic Vision Datasets: Similarities and Differences. *arXiv preprint arXiv:2005.02183*, 2020. 7, 10

[24] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005. 2

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 4

[26] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829*, 2015. 2, 7

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 3

[28] Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*, pages 7005–7015, 2018. 2, 7

[29] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *Frontiers in Neuroscience*, 14:424, 2020. 2, 7, 10

[30] David Kappel, Robert Legenstein, Stefan Habenschuss, Michael Hsieh, and Wolfgang Maass. A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *eNeuro*, 5(2), 2018. 2

[31] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018. 2

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 10

[33] Christof Koch, Moshe Rapp, and Idan Segev. A brief history of time (constants). *Cerebral Cortex*, 6(2):93–101, 1996. 2

[34] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. 2

[35] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2

[36] Eimantas Ledinauskas, Julius Ruseckas, Alfonsas Juršėnas, and Giedrius Buračas. Training Deep Spiking Neural Networks. *arXiv preprint arXiv:2006.04436*, 2020. 3, 4

[37] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020. 2, 5, 6, 7, 9, 11

[38] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016. 2, 7, 10

[39] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017. 2

[40] Qianhui Liu, Haibo Ruan, Dong Xing, Huajin Tang, and Gang Pan. Effective AER Object Classification Using Segmented Probability-Maximization Learning in Spiking Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1308–1315, 2020. 7

[41] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 10

[42] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. 1

[43] Maurizio Mattia and Paolo Del Giudice. Population dynamics of interacting spiking neurons. *Physical Review E*, 66(5):051917, 2002. 2

[44] Ammar Mohemmed, Stefan Schliebs, Satoshi Matsuda, and Nikola Kasabov. Span: Spike pattern association neuron for learning spatio-temporal spike patterns. *International Journal of Neural Systems*, 22(04):1250012, 2012. 2

[45] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, Simon J Thorpe, and Timothée Masquelier. Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognition*, 94:87–95, 2019. 2

[46] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine*, 36:61–63, 2019. 2

[47] Daniel Neil and Shih-Chii Liu. Effective sensor fusion with event-based sensors and deep network architectures. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2282–2285. IEEE, 2016. 10

[48] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences, 2016. 10

[49] Bernhard Nessler, Michael Pfeiffer, Lars Buesing, and Wolfgang Maass. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Computational Biology*, 9(4):e1003037, 2013. 2

[50] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015. 2

[51] Filip Ponulak and Andrzej Kasiński. Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. *Neural Computation*, 22(2):467–510, 2010. 2

[52] Wiebke Potjans, Markus Diesmann, and Abigail Morrison. An imperfect dopaminergic error signal can drive temporal-difference learning. *PLoS Computational Biology*, 7(5):e1001133, 2011. 2

[53] Nitin Rathi and Kaushik Roy. DIET-SNN: Direct Input Encoding With Leakage and Threshold Optimization in Deep Spiking Neural Networks. *arXiv preprint arXiv:2008.03658*, 2020. 2, 4, 5, 7

[54] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*, 2020. 5

[55] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. 1

[56] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017. 2, 5, 6, 7

[57] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019. 2, 5, 7

[58] H Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40(6):1063–1073, 2003. 2

[59] Karthik H Shankar and Marc W Howard. A scale-invariant internal representation of time. *Neural Computation*, 24(1):134–193, 2012. 2

[60] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in Neural Information Processing Systems*, 31:1412–1421, 2018. 2, 5, 7, 11

[61] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1731–1740, 2018. 7

[62] Narayan Srinivasa and Youngkwan Cho. Self-organizing spiking neural model for learning fault-tolerant spatio-motor transformations. *IEEE Transactions on Neural Networks and Learning Systems*, 23(10):1526–1538, 2012. 2

[63] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018. 2, 7, 10

[64] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1311–1318, 2019. 2, 3, 5, 7, 9, 10

[65] Yujie Wu, Rong Zhao, Jun Zhu, Feng Chen, Mingkun Xu, Guoqi Li, Sen Song, Lei Deng, Guanrui Wang, Hao Zheng, et al. Brain-inspired global-local hybrid learning towards human-like intelligence. *arXiv preprint arXiv:2006.03226*, 2020. 10

[66] Simei Gomes Wysoski, Lubica Benuskova, and Nikola Kasabov. Fast and adaptive network of spiking neurons for multi-view visual pattern recognition. *Neurocomputing*, 71(13):2563–2575, 2008. Artificial Neural Networks (ICANN 2006) / Engineering of Intelligent Systems (ICEIS 2006). 11

[67] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 2

[68] Yannan Xing, Gaetano Di Caterina, and John Soraghan. A new spiking convolutional recurrent neural network (scrnn) with applications to event-based hand gesture recognition. *Frontiers in Neuroscience*, 14:1143, 2020. 7, 10

[69] Bojian Yin, Federico Corradi, and Sander M Bohté. Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks. *arXiv preprint arXiv:2005.11633*, 2020. 2, 4

[70] Qiang Yu, Kay Chen Tan, and Huajin Tang. Pattern recognition computation in a spiking neural network with temporal encoding and learning. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2012. 11

[71] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haizhou Li. Rapid feedforward computation by temporal encoding and learning with spiking neurons. *IEEE Transactions on Neural Networks and Learning Systems*, 24(10):1539–1552, 2013. 11

[72] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haoyong Yu. A brain-inspired spiking neural network model with temporal encoding and learning. *Neurocomputing*, 138:3–13, 2014. 11

[73] Mengwen Yuan, Xi Wu, Rui Yan, and Huajin Tang. Reinforcement Learning in Spiking Neural Networks with Stochastic and Deterministic Synapses. *Neural Computation*, 31(12):2368–2389, 2019. 2

[74] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *BioRxiv*, 2020. 2, 10

[75] Wenrui Zhang and Peng Li. Spike-train level backpropagation for training deep recurrent spiking neural networks. In *Advances in Neural Information Processing Systems*, pages 7802–7813, 2019. 2, 5, 7, 11

[76] Romain Zimmer, Thomas Pellegrini, Srisht Fateh Singh, and Timothée Masquelier. Technical report: supervised training of convolutional spiking neural networks with PyTorch. *arXiv preprint arXiv:1911.10124*, 2019. 2