Training High-Performance Low-Latency Spiking Neural Networks by Differentiation on Spike Representation

Anonymous CVPR submission

Paper ID 10529

Abstract

Spiking Neural Network (SNN) is a promising energy-efficient AI model when implemented on neuromorphic hardware. However, it is a challenge to efficiently train SNNs due to their non-differentiability. Most existing methods either suffer from high latency (i.e. long simulation time), or cannot achieve as high performance as Artificial Neural Networks (ANNs). In this paper, we propose the Differentiation on Spike Representation (DSR) method, which could achieve high performance that is competitive to ANNs yet with low latency. First, we encode the spike trains generated by the SNN into spike representation using (weighted) firing rate coding. Based on the spike representation, we systematically derive that the spiking dynamics with common neural models can be represented as some sub-differentiable mapping. With this viewpoint, our proposed DSR method trains SNNs through gradients of the mapping and avoids the common non-differentiability problem in SNN training. Then we analyze the error when representing the specific mapping with the forward computation of the SNN. To reduce such error, we propose to train the spike threshold in each layer, and to introduce a new hyperparameter for the neural models. With these components, the DSR method can achieve state-of-the-art SNN performance with low latency on both static and neuromorphic datasets, including CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10.

1. Introduction

Inspired by biological neurons that communicate using spikes, Spiking Neural Networks (SNNs) have recently received surging attention. This promise depends on their energy efficiency on neuromorphic hardware [7,26,30], while deep Artificial Neural Networks (ANNs) require substantial power consumption.

However, the training of SNNs is a major challenge [38] since information in SNNs is transmitted through non-differentiable spike trains. Specifically, the non-differentiability in SNN computation hampers the effective usage of gradient-based backpropagation methods. To tackle this problem, the surrogate gradient (SG) method [12,28,35,41,48] and the ANN-to-SNN conversion method [4, 9, 33, 34, 45] have been proposed and yielded the best performance. In the SG method, an SNN is regarded as a recurrent neural network (RNN) and trained by the backpropagation through time (BPTT) framework. And during backpropagation, gradients of non-differentiable spike functions are approximated by some surrogate gradients. Although the SG method can train SNNs with low latency (*i.e.* short simulation time), it cannot achieve high performance comparable to leading ANNs. Besides, the adopted BPTT framework needs to backpropagate gradients through both the layer-by-layer spatial domain and the temporal domain, leading to a long training time and high memory cost of the SG method. The high training costs further limit the usage of large-scale network architectures. On the other hand, the ANN-to-SNN conversion method directly determines the network weights of an SNN from a corresponding ANN, relying on the connection between firing rates of the SNN and activations of the ANN. The conversion method enables the obtained SNN to perform as competent as its ANN counterpart. However, intolerably high latency is typically required, since only a large number of time steps can make the firing rates closely approach the high-precision activation values of ANNs [18, 33]. Overall, SNNs obtained by the two wildly-used methods either cannot compete their ANN counterparts, or suffer from high latency.

In this paper, we overcome both the low performance and high latency issues by introducing the Differentiation on Spike Representation (DSR) method to train SNNs. First, we treat the (weighted) firing rate of the spiking neurons as spike representation. Based on the representation, we show that the forward computation of an SNN with common spiking neurons can be represented as some sub-differentiable mapping. We then derive the backpropagation algorithm while treating the spike representation as the information carrier. In this way, our method does not require backpropagation through the temporal domain, thus avoiding the common non-differentiability problem in SNN training. To ef-

Table 1. Comparison of the ANN-to-SNN conversion, surrogate gradient (SG), and proposed DSR method with respect to latency, performance with low latency, training cost, and applicability on neuromorphic data.

	Conversion	SG	DSR
Latency	High	**Low**	**Low**
Performace w/ Low Latency	Low	Medium	**High**
Training Cost	**Low**	High	Medium
Neuromorphic Data	Non-applicable	**Applicable**	**Applicable**

fectively train SNNs with low latency, we further study the representation error due to the SNN-to-mapping approximation, and propose to train the spike thresholds and introduce a new hyperparameter for the spiking neural models to reduce the error. With these methods, we can train high-performance low-latency SNNs. And the comparison of the properties between the DSR method and other methods is illustrated in Tab. 1. Formally, our main contributions are summarized as follows:

1. We systematically study the spike representation for common spiking neural models, and propose the DSR method that uses the representation to train SNNs by backpropagation. The proposed method avoids the non-differentiability problem in SNN training and does not require the costly error backpropagation through the temporal domain.

2. We propose to train the spike thresholds and introduce a new hyperparameter for the spiking neural models to reduce the representation error. The two techniques greatly help the DSR method to train SNNs with high performance and low latency.

3. Our model achieves competitive or state-of-the-art SNN performance with low latency on the CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10 classification tasks. Furthermore, the experiments also prove the effectiveness of the DSR method under ultra-low latency or deep network structures.

2. Related Work

Many works seek biological plausibility in training SNNs [5, 21, 24] using derivations of the Hebbian learning rule [17]. However, this method cannot achieve competitive performance and cannot be applicable on complicated datasets. Besides the brain-inspired method, SNN learning methods can be mainly categorized into two classes: ANN-to-SNN conversion [9–11,14,15,20,22,33,34,45] and direct training [1, 2, 12, 19, 27, 28, 35, 41, 42, 44, 46–48]. We discuss both the conversion and direct training method, then analyze the information representation used in them.

ANN-to-SNN Conversion The feasibility of this conversion method relies on the fact that the firing rates of an SNN can be estimated by activations of an ANN with corresponding architecture and weights [33]. With this method, the parameters of a target SNN are directly determined from a source ANN. And the performance of the target SNN is supposed to be not much worse than the source ANN. Many effective techniques have been proposed to reduce the performance gap, such as weight normalization [34], temporal switch coding [14], rate norm layer [11], and bias shift [9]. Recently, the conversion method has achieved high-performance ANN-to-SNN conversion [9, 25, 45], even on ImageNet. However, the good performance is at the expense of high latency, since only high latency can make the firing rates closely approach the high-precision activation. This fact hurts the energy efficiency of SNNs when using the conversion method. Furthermore, the conversion method is not suitable for neuromorphic data. In this paper, we borrow the idea of ANN-SNN mapping to design the backpropagation algorithm for training SNNs. However, unlike usual ANN-to-SNN conversion methods, the proposed DSR method can obtain high performance with low latency on both static and neuromorphic data.

Direct Training Inspired by the immense success of gradient descent-based algorithms for training ANN, some works regard an SNN as an RNN and directly train it with the BPTT method. This scheme typically leverages surrogate gradient to deal with the discontinuous spike functions [2,28,41,48], or calculate the gradients of loss with respect to spike times [27,43,49]. Among them, the surrogate gradient method achieves better performance with lower latency [12, 48]. However, those approaches need to backpropagate error signals through time steps and thus suffer from high computational costs during training [8]. Furthermore, the inaccurate approximations for computing the gradients or the "dead neuron" problem [35] limit the training effect and the use of large-scale network architectures. The proposed method uses spike representation to calculate the gradients of loss and need not backpropagate error through time steps. Therefore, the proposed method avoids the common problems for direct training. A few works [39, 40] also use the similar idea of decoupling the forward and backward passes to train feedforward SNNs; however, they neither systematically analyze the representation schemes nor the representation error, and they cannot achieve comparable accuracy as ours, even with high latency.

Information Representation in SNNs In SNNs, the information is carried by some representation of spike trains [29]. There are mainly two information representation schemes: temporal coding and rate coding. These two schemes treat exact firing times and firing rates, respectively, as the information carrier. Temporal coding is adopted by some direct training methods that calculate gradients with respect to spike times [27, 43, 49], or few ANN-to-SNN methods [14, 37]. With temporal coding, those methods typically enjoy low energy consumption on neuromorphic chips due to sparse spikes. However, those methods either require chip-unfriendly neuron settings [14, 37, 49], or only perform well on simple datasets. Rate coding is adopted by most ANN-to-SNN methods [9–11, 15, 22, 33, 34, 45] and many direct training methods [12, 28, 35, 39–42, 44, 48]. The rate coding-based methods typically achieve better performances than those with temporal coding. Furthermore, recent progress shows the potential of rate-coding based methods on low latency or sparse firing [12, 44, 48], making it possible to reach the same or even better level of energy efficiency as temporal coding scheme. In this paper, we adopt the rate coding scheme to train SNNs for the sake of both high performance and low latency.

3. Proposed Differentiation on Spike Representation (DSR) Method

3.1. Spiking Neural Models

Spiking neurons imitate the biological neurons that communicate with each other by spike trains. In this paper, we adopt the widely used integrate-and-fire (IF) model and leaky-integrate-and-fire (LIF) model [3], both of which are simplified models to characterize the process of spike generation. Each IF neuron or LIF neuron 'integrates' the received spike as its membrane potential $V(t)$, and the dynamics of membrane potential can be formally depicted as

$$\text{IF:} \quad \frac{\mathrm{d}V(t)}{\mathrm{d}t} = I(t), \qquad V < V_{th}, \tag{1}$$

$$\text{LIF:} \quad \tau\frac{\mathrm{d}V(t)}{\mathrm{d}t} = -(V(t) - V_{rest}) + I(t), \quad V < V_{th}, \tag{2}$$

where V_{rest} is the resting potential, τ is the time constant, V_{th} is the spike threshold, and I is the input current which is related to received spikes. Once the membrane potential V exceeds the predefined threshold V_{th} at time t_f, the neuron will fire a spike and reset its membrane potential to the resting potential V_{rest}. The output spike train can be expressed using the Dirac delta function $s(t) = \sum_{t_f} \delta(t - t_f)$.

In practice, discretization for the dynamics is required. The discretized model is described as:

$$\begin{cases} U[n] = f(V[n-1], I[n]), & \text{(3a)} \\ s[n] = H(U[n] - V_{th}), & \text{(3b)} \\ V[n] = U[n] - V_{th}s[n], & \text{(3c)} \end{cases}$$

where $U[n]$ is the membrane potential before resetting, $s[n] \in \{0, 1\}$ is the output spike, $H(x)$ is the Heaviside step function, and f is the membrane potential update function. In the discretization, both $V[0]$ and V_{rest} are set to be 0 for simplicity, and therefore $V_{th} > 0$. The function $f(\cdot, \cdot)$ for IF and LIF models can be described as:

$$\text{IF: } f(V, I) = V + I, \tag{4}$$

$$\text{LIF: } f(V, I) = e^{-\frac{\Delta t}{\tau}} V + \left(1 - e^{-\frac{\Delta t}{\tau}}\right) I, \tag{5}$$

where $\Delta t < \tau$ is the discrete step for LIF model. In practice, we set Δt to be much less that τ. Different from other literature [9, 13, 41], we explicitly introduce the hyperparameter Δt to ensure a large feasible region for τ, since the discretization for LIF model is only valid when the discrete step $\tau > \Delta t$ [13]. For example, $\tau = 1$ is allowed in our setting, while some other works prohibit it since they set $\Delta t = 1$. We use the "reduce by subtraction" method [40,44] for resetting the membrane potential in Eq. (3c). Combing Eqs. (3a) and (3c), we get a more concise update rule for the membrane potential:

$$\mathbf{V}^i[n] = f(V[n-1], I[n]) - V_{th}s[n]. \tag{6}$$

Eq. (6) is used to define the forward pass of SNNs.

3.2. Forward Pass

In this paper, we consider L-layer feedforward SNNs with the IF or LIF models. According to Eqs. (4) and (6), the spiking dynamics for an SNN with the IF model can be described as:

$$\mathbf{V}^i[n] = \mathbf{V}^i[n-1] + V_{th}^{i-1}\mathbf{W}^i\mathbf{s}^{i-1}[n] - V_{th}^i\mathbf{s}^i[n], \tag{7}$$

where $i = 1, 2, \cdots, L$ is the layer index, $n \in [N]$ is the time step index and N is the total number of time steps, $\mathbf{s}^0$ are the input data to the network, $\mathbf{s}^i$ are the output spike trains of the i^{th} layer for $i = 1, 2, \cdots, L$, and $\mathbf{W}^i$ are trainable synaptic weights from the $(i-1)^{\text{th}}$ layer to the i^{th} layer. Spikes are generated according to Eq. (3b), and the weighted sums of the $(i-1)^{\text{th}}$ layer's spikes are treated as input currents to the i^{th} layer. Furthermore, the spike thresholds are the same for all IF neurons in one particular layer. Similarly, according to Eqs. (5) and (6), the spiking dynamics for an SNN with the LIF model can be shown as:

$$\begin{aligned} \mathbf{V}^i[n] &= \exp(-\frac{\Delta t}{\tau^i})\mathbf{V}^i[n-1] \\ &+ (1 - \exp(-\frac{\Delta t}{\tau^i}))\Delta t V_{th}^{i-1}\mathbf{W}^i\mathbf{s}^{i-1}[n] - V_{th}^i\mathbf{s}^i[n], \end{aligned} \tag{8}$$

where Δt is set to be a positive number much less than τ_i, and it appears to simplify the analysis on spike representation schemes in Sec. 3.3. In Eqs. (7) and (8), we only consider fully connected layers. However, other neural network components like convolution, skip connection, and average pooling can also be adopted.

The input $\mathbf{s}^0$ to the SNN can be both neuromorphic data or static data (*e.g.*, images). While neuromorphic data are naturally adapted to SNNs, for static data, we repeatedly

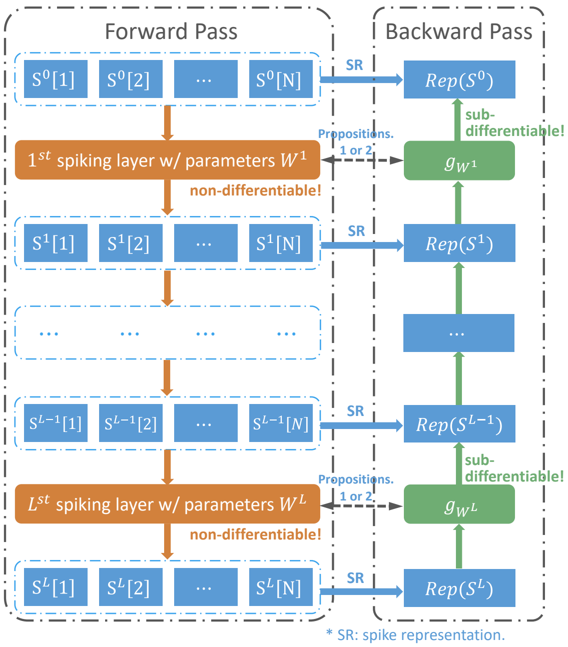


Figure 1. The pipeline of proposed DSR method. The left part shows the forward computation of an SNN. The right shows the error backpropagation through the sub-differentiable mapping $g_{\mathbf{W}^i}$.

apply them to the first layer at each time step [33, 44, 48]. With this method, the first layer can be treated as the spike-train data generator. We use the spike trains $\mathbf{s}^L$ as the output data of the SNN, whose setting is more biologically plausible than prohibiting firing for the last layer and using the membrane potentials as the network output [23, 39].

3.3. Spike Representation

In this subsection, we show that the forward computation for each layer of an SNN with the IF or LIF neurons can be represented as a sub-differentiable mapping using spike representation as the information carrier. And the spike representation is obtained by (weighted) firing rate coding. Specifically, denoting by $\mathbf{s}^i$ the output spike trains of the i^{th} layer, the relationship between the SNN and the mapping can be expressed as

$$\mathrm{Rep}(\mathbf{s}^i) \approx g_{\mathbf{W}^i}(\mathrm{Rep}(\mathbf{s}^{i-1})),\ i = 1, 2, \cdots, L, \tag{9}$$

where $\mathrm{Rep}(\mathbf{s}^i)$ is the spike representation of $\mathbf{s}^i$, $\mathbf{W}^i$ are the SNN parameters for the i^{th} layer, and $g_{\mathbf{W}^i}$ is the sub-differentiable mapping also parameterized by $\mathbf{W}^i$. Then the SNN parameters $\mathbf{W}^i$ can be learned through gradients of $g_{\mathbf{W}^i}$. The illustration of the SNN-to-mapping representation is shown in Fig. 1.

We first use weighted firing rate coding to derive the formulae of spike representation $\mathrm{Rep}(\mathbf{s}^i)$ and the sub-differentiable mapping $g_{\mathbf{W}^i}$ for the LIF model. Then we briefly introduce the formulae for the IF model, which are simple extensions of those for the LIF model. Training SNNs based on the spike representation schemes is described in Sec. 3.4.

3.3.1 Spike Representation for the LIF Model

We first consider the LIF model defined by Eqs. (3a) to (3c) and (5). To simplify the notation, define $\lambda = \exp(-\frac{\Delta t}{\tau})$. We further define $\hat{I}[N] = \frac{\sum_{n=1}^{N} \lambda^{N-n} I[n]}{\sum_{n=1}^{N} \lambda^{N-n}}$ as the weighted average input current until the time step N, and define $\hat{a}[N] = \frac{V_{th} \sum_{n=1}^{N} \lambda^{N-n} s[n]}{\sum_{n=1}^{N} \lambda^{N-n} \Delta t}$ as the scaled weighted firing rate. Here we treat $\hat{a}[N]$ as the spike representation of the spike train $\{s[n]\}_{n=1}^{N}$ for the LIF model. The key idea is to directly determine the relationship between $\hat{I}[N]$ and $\hat{a}[N]$ using a (sub-)differentiable mapping.

In detail, combining Eqs. (5) and (6), and multiplying the combined equation by λ^{N-n}, we have

$$\begin{aligned} \lambda^{N-n} V[n] = \lambda^{N-n+1} V[n-1] \\ + (1-\lambda)\lambda^{N-n} I[n] - \lambda^{N-n} V_{th} s[n]. \end{aligned} \tag{10}$$

Summing Eq. (10) over $n = 1$ to N, we can get

$$V[N] = (1-\lambda) \sum_{n=1}^{N} \lambda^{N-n} I[n] - \sum_{n=1}^{N} \lambda^{N-n} V_{th} s[n]. \tag{11}$$

Dividing Eq. (11) by $\Delta t \sum_{n=1}^{N} \lambda^{N-n}$ and then rearrange the terms, we have

$$\hat{a}[N] = \frac{(1-\lambda)\hat{I}[N]}{\Delta t} - \frac{V[N]}{\Delta t \sum_{n=1}^{N} \lambda^{N-n}}. \tag{12}$$

Note that we can further approximate $\frac{1-\lambda}{\Delta t}$ in Eq. (12) by $\frac{1}{\tau}$, since $\lim\limits_{\Delta t \to 0} \frac{1-\lambda}{\Delta t} = \frac{1}{\tau}$ and we set $\Delta t \ll \tau$. Then we have

$$\hat{a}[N] \approx \frac{\hat{I}[N]}{\tau} - \frac{V[N]}{\Delta t \sum_{n=1}^{N} \lambda^{N-n}}. \tag{13}$$

Eq. (13) is the basic formula to determine the mapping from $\hat{I}[N]$ to $\hat{a}[N]$. Note that in Eq. (13), the term $\frac{V[N]}{\Delta t \sum_{n=1}^{N} \lambda^{N-n}}$ cannot be directly determined only given $\hat{I}[N]$. However, taking $\hat{a}[N] \in [0, \frac{V_{th}}{\Delta t}]$ into consideration and assuming V_{th} is small, we can ignore the term $\frac{V[N]}{\Delta t \sum_{n=1}^{N} \lambda^{N-n}}$ in Eq. (13), and further approximate $\hat{a}[N]$ as

$$\lim_{N \to \infty} \hat{a}[N] \approx \mathrm{clamp}\left(\lim_{N \to \infty} \frac{\hat{I}[N]}{\tau}, 0, \frac{V_{th}}{\Delta t} \right), \tag{14}$$

where the clamp function is defined as $\mathrm{clamp}(x, a, b) = \max(a, \min(x, b))$. Detailed derivation and mild assumptions for Eq. (14) are shown in the Supplementary Materials. Applying Eq. (14) to feedforward SNNs with multiple LIF neurons, we have Proposition 1, which is used to train SNNs.

Proposition 1. *Consider an SNN with LIF neurons defined by Eq. (8). Define* $\hat{\mathbf{a}}^0[N] = \frac{\sum_{n=1}^{N} \lambda_i^{N-n} \mathbf{s}^0[n]}{\sum_{n=1}^{N} \lambda_i^{N-n} \Delta t}$ *and* $\hat{\mathbf{a}}^i[N] = \frac{V_{th}^i \sum_{n=1}^{N} \lambda_i^{N-n} \mathbf{s}^i[n]}{\sum_{n=1}^{N} \lambda_i^{N-n} \Delta t}, \forall i = 1, 2, \cdots, L$, *where* $\lambda_i = \exp(-\frac{\Delta t}{\tau^i})$. *Further define sub-differentiable mappings*

$$\mathbf{z}^i = \text{clamp}\left(\frac{1}{\tau^i}\mathbf{W}^i\mathbf{z}^{i-1}, 0, \frac{V_{th}^i}{\Delta t}\right), i = 1, 2, \cdots, L.$$

If $\lim_{N\to\infty} \hat{\mathbf{a}}^i[N] = \mathbf{z}^i$ *for* $i = 0, 1, \cdots, L-1$, *then* $\hat{\mathbf{a}}^{i+1}[N]$ *approximates* $\mathbf{z}^{i+1}$ *when* $N \to \infty$.

3.3.2 Spike Representation for the IF Model

We then consider the IF model defined by Eqs. (3a) to (3c) and (4). Define $\bar{I}[N] = \frac{1}{N}\sum_{n=1}^{N} I[n]$ as the average input current until the time step N, and define $a[N] = \frac{1}{N}\sum_{n=1}^{N} V_{th} s[n]$ as the scaled firing rate. We then treat $a[N]$ as the spike representation of the spike train $\{s[n]\}_{n=1}^{N}$ for the IF model. We can use similar arguments shown in Sec. 3.3.1 to determine the relationship between $\hat{I}[N]$ and $\hat{a}[N]$ as

$$\lim_{N\to\infty} a[N] = \text{clamp}\left(\lim_{N\to\infty} \bar{I}[N], 0, V_{th}\right). \tag{15}$$

Detailed assumptions and derivation for Eq. (15) are shown in the Supplementary Materials. With Eq. (15), we can have Proposition 2 to train feedforward SNNs with the IF model.

Proposition 2. *Consider an SNN with IF neurons defined by Eq. (7). Define* $\mathbf{a}^0[N] = \frac{1}{N}\sum_{n=1}^{N} \mathbf{s}^0[n]$ *and* $\mathbf{a}^i[N] = \frac{1}{N}\sum_{n=1}^{N} V_{th}^L \mathbf{s}^i[n], \forall i = 1, 2, \cdots, L$. *Further define sub-differentiable mappings:*

$$\mathbf{z}^i = \text{clamp}\left(\mathbf{W}^i\mathbf{z}^{i-1}, 0, V_{th}^i\right), i = 1, 2, \cdots, L.$$

If $\lim_{N\to\infty} \mathbf{a}^0[N] = \mathbf{z}^0$, *then* $\lim_{N\to\infty} \mathbf{a}^i[N] = \mathbf{z}^i$.

3.4. Differentiation on Spike Representation

In this subsection, we use the spike representation for the spiking neural models to drive the backpropagation training algorithm for SNNs, based on Propositions 1 and 2. And the illustration can be found in Fig. 1.

Define the spike representation operator $\text{r}(\cdot)$ with spike train $s = (s[1], \cdots, s[N])$ as input, such that $\text{r}(s) = \frac{1}{N}\sum_{n=1}^{N} V_{th} s[n]$ for the IF model, and $\text{r}(s) = \frac{V_{th}\sum_{n=1}^{N}\lambda^{N-n} s[n]}{\sum_{n=1}^{N}\lambda^{N-n}\Delta t}$ for the LIF model, where $\lambda = \exp(-\frac{\Delta t}{\tau})$. With the spike representation, we define the final output of the SNN as $\mathbf{o}^L = \text{r}(\mathbf{s}^L)$, where $\mathbf{s}^L$ is the output spike trains from the last layer and $\text{r}(\cdot)$ is defined element-wise. We use cross-entropy as the loss function ℓ.

The proposed DSR method backpropagates the gradient of error signals based on the representation of spike trains in each layer, $\mathbf{o}^i = \text{r}(\mathbf{s}^i)$, where $i = 1, 2, \cdots, L$ is the layer index. By applying chain rule, the required gradient $\frac{\partial \ell}{\partial \mathbf{W}^i}$ can be computed as

$$\frac{\partial \ell}{\partial \mathbf{W}^i} = \frac{\partial \ell}{\partial \mathbf{o}^i}\frac{\partial \mathbf{o}^i}{\partial \mathbf{W}^i}, \quad \frac{\partial \ell}{\partial \mathbf{o}^i} = \frac{\partial \ell}{\partial \mathbf{o}^{i+1}}\frac{\partial \mathbf{o}^{i+1}}{\partial \mathbf{o}^i}, \tag{16}$$

where $\frac{\partial \mathbf{o}^{i+1}}{\partial \mathbf{o}^i}$ and $\frac{\partial \mathbf{o}^i}{\partial \mathbf{W}^i}$ can be computed with Propositions 1 and 2. Specifically, from Sec. 3.3, we have

$$\mathbf{o}^i = \text{r}(\mathbf{s}^i) \approx \text{clamp}(\mathbf{W}^i\,\text{r}(\mathbf{s}^{i-1}), 0, b_i), \; i = 1, 2, \cdots, L, \tag{17}$$

where $b_i = V_{th}^i$ for the IF model, and $b_i = \frac{V_{th}^i}{\Delta t}$ for the LIF model. Therefore, we can calculate $\frac{\partial \mathbf{o}^{i+1}}{\partial \mathbf{o}^i}$ and $\frac{\partial \mathbf{o}^i}{\partial \mathbf{W}^i}$ based on Eq. (17). The pseudocode of the proposed DSR method can be found in the Supplementary Materials.

With the proposed DSR method, we avoid two common problems in SNN training. First, this method does not require backpropagation through the temporal domain, improving the training efficiency when compared with the BPTT type methods, especially when the number of time steps is not ultra-small. Second, this method does not need to handle the non-differentiability of spike functions, since the signals are backpropagated through sub-differentiable mapping. Although there exists representation error due to finite time steps, we can reduce it, as described in Sec. 4.

4. Reducing Representation Error

Propositions 1 and 2 show that the (weighted) firing rate can gradually estimate or converge to the output of a sub-differentiable mapping. And Sec. 3.4 shows that we can train SNNs by backpropagation using spike representation. However, in practice we want to simulate SNNs with only a small number of time steps, for the sake of low energy consumption. The low latency will further introduce representation error that hinders effective training. In this subsection, we study the representation error and propose to train the spike threshold and introduce a new hyperparameter for the neural models to reduce the error.

The representation error e_r can be decomposed as $e_r = e_q + e_d$, where e_q is the "quantization error" and e_d is the "deviation error". The quantization error e_q exists due to the imperfect precision of the firing rate, when assuming the same input currents at all time steps. For example, it can only take value in the form $\frac{n}{N}$ for the IF neuron, where $n \in \mathbb{N}$ and N is the number of time steps. And the deviation error e_q exists due to the inconsistency of input currents at different time steps. For example, when the average input current is 0, the output firing rate is supposed to be 0; however, it can be significantly larger than 0 if the input currents are positive during the beginning time steps.

From the statistical perspective, the expectation for e_d is 0, assuming i.i.d. input currents at different time steps. Therefore, the "deviation error" e_d will not affect training

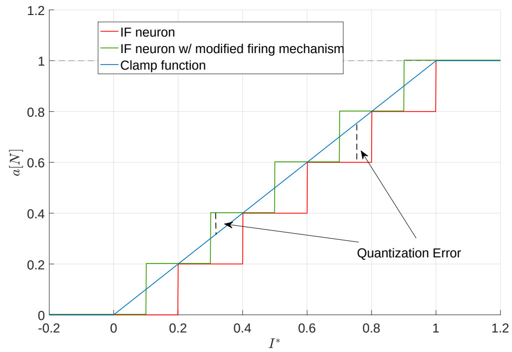


Figure 2. The firing rate of the IF neuron approximates the clamp function given unchanged input currents. Spike generating of IF neurons w/ or w/o introducing a new hyperparameter for spiking neurons is controlled by Eqs. (3b) and (20), respectively. Here we set the threshold $\theta = 1$ and the time-steps $N = 5$.

too much when using stochastic optimization algorithms. Next, we dig into e_q with the IF model and then propose methods to reduce it. And similar arguments can be derived for the LIF model. When given unchanged input currents I^* at all time steps to the IF neuron, the scaled average firing rate $a[N] = \frac{1}{N}\sum_{n=1}^{N} V_{th}s[n]$ can be determined as

$$a[N] = \frac{V_{th}}{N} \cdot \text{clamp}\left(\left\lfloor \frac{NI^*}{\theta} \right\rfloor, 0, N\right), \tag{18}$$

shown as the red curve in Fig. 2, where $\lfloor\cdot\rfloor$ is the floor rounding operator. Inspired by Eq. (18), we propose two methods to reduce the quantization error.

Training the Spike Threshold From Fig. 2 and Eq. (18), we observe that using small spike thresholds can reduce the quantization error. However, it also weakens the approximation capacity of the SNNs, since the scaled (weighted) firing rate will be in a small range then. Inspired by activation clipping methods for training quantized neural networks [6], in this paper, we treat the spike threshold of each layer as parameters to be trained, and include an L2-regularizer for the thresholds in the loss function to balance the tradeoff between quantization error and approximation capacity. To train the spike thresholds using backpropagation, we calculate the gradients with respect to them based on the spike representation introduced in Sec. 3.3. For example, using Eq. (15), for one IF neuron with average input current I^* and steady scaled firing rate a^*, we have

$$\frac{\partial a^*}{\partial V_{th}} = \begin{cases} 1, & \text{if } I^* > V_{th}, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Then we can calculate the gradient of the loss function with respect to the threshold by the chain rule. A similar calculation applies to LIF neurons. In practice, since we use mini-batch optimization methods to train SNNs, the gradient for each threshold is proportional to the batch size by the chain rule. Thus, we scale the gradient regarding different batch sizes and spiking neural models.

Introducing a new hyperparameter for the neural models We can introduce a new hyperparameter for spiking neurons to control the neuron firing to reduce the quantization error. Formally, we change Eq. (3b) to

$$s[n] = H(U[n] - \alpha V_{th}) \tag{20}$$

to get a new firing mechanism, where $\alpha \in [0, 1]$ is a hyperparameter. For the IF model with the new firing mechanism and $\alpha = 0.5$, using the same notation as in Eq. (18), the scaled firing rate becomes

$$a[N] = \frac{V_{th}}{N} \cdot \text{clamp}\left(\left[\frac{NI^*}{\theta} \right], 0, N\right), \tag{21}$$

shown as the green curve in Fig. 2, where $[\cdot]$ is the rounding operator. From Fig. 2, we can see that the maximum absolute quantization error is halved when using this mechanism. Furthermore, since $\alpha = 0.5$ makes the average absolute quantization error minimized, $\alpha = 0.5$ is the best choice for the IF model. On the other hand, for the LIF model, the best choice for α changes when setting different latency N, so we choose different α in our experiments to minimize the average absolute quantization error.

5. Experiments

We first evaluate the proposed DSR method and compare it with other works on visual object recognition benchmarks, CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10. We then demonstrate the effectiveness of our method when the number of time steps becomes smaller and smaller, or the network becomes deeper and deeper. We also test the effectiveness of the methods to reduce representation error. Please refer to the Supplementary Materials for experiment details, including dataset description, training parameters, network architectures, and used Batch Normalization component.

5.1. Comparison to the State-of-the-Art

The comparison on CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10 is shown in Tab. 2.

For the CIFAR-10 and the CIFAR-100 datasets, we use pre-activation ResNet-18 [16] as the network architecture. In order to adapt the architecture to an SNN, we add spiking neuron layers after the pooling layer and the last fully connected layer. Tab. 2 shows that the proposed DSR method outperforms all other methods with 20 time steps for both the IF and the LIF models, based on 3 runs of experiments. Especially, our method achieves accuracies that are 5%-10% higher on CIFAR-100 when compared to others. Furthermore, the obtained SNNs have similar or even better performance compared to ANNs with the same network

Table 2. Performance on CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10. For the first three datasets, we categorize the methods into 4 classes: ANN, the ANN-to-SNN method, the direct training method, and our proposed method. Different types of methods are separated by horizontal lines. We bold the best result for the LIF model, and underline the best result for the IF model.

	Method	Network	Neural Model	Time-steps	Accuracy
CIFAR-10	ANN [1]	PreAct-ResNet-18	/	/	95.41%
	ANN-to-SNN [9]	ResNet-20	IF	128	93.56%
	ANN-to-SNN [15]	VGG-16	IF	2048	93.63%
	ANN-to-SNN [45]	VGG-like	IF	600	94.20%
	Tandem Learning [40]	CIFARNet	IF	8	90.98%
	ASF-BP [39]	VGG-7	IF	400	91.35%
	STBP [41]	CIFARNet	LIF	12	90.53%
	IDE [44]	CIFARNet-F	LIF	100	$92.52\% \pm 0.17\%$
	STBP-tdBN [48]	ResNet-19	LIF	6	93.16%
	TSSL-BP [47]	CIFARNet	LIF w/ synaptic model	5	91.41%
	DSR (ours)	PreAct-ResNet-18	IF	20	$\underline{95.24\% \pm 0.17\%}$
		PreAct-ResNet-18	LIF	20	$\mathbf{95.40\% \pm 0.15\%}$
CIFAR-100	ANN [1]	PreAct-ResNet-18	/	/	78.12%
	ANN-to-SNN [9]	ResNet-20	IF	400-600	69.82%
	ANN-to-SNN [15]	VGG-16	IF	768	70.09%
	ANN-to-SNN [45]	VGG-like	IF	300	71.84%
	Hybrid Training [32]	VGG-11	LIF	125	67.84%
	DIET-SNN [31]	VGG-16	LIF	5	69.67%
	IDE [44]	CIFARNet-F	LIF	100	$73.07\% \pm 0.21\%$
	DSR (ours)	PreAct-ResNet-18	IF	20	$\underline{78.20\% \pm 0.13\%}$
		PreAct-ResNet-18	LIF	20	$\mathbf{78.50\% \pm 0.12\%}$
ImageNet	ANN [1]	PreAct-ResNet-18	/	/	70.79%
	ANN-to-SNN [34]	ResNet-34	IF	2000	65.47%
	ANN-to-SNN [15]	ResNet-34	IF	4096	$\underline{69.89\%}$
	Hybrid training [32]	ResNet-34	LIF	250	61.48%
	STBP-tdBN [48]	ResNet-34	LIF	6	63.72%
	SEW ResNet [12]	SEW ResNet-34	LIF	4	67.04%
	SEW ResNet [12]	SEW ResNet-18	LIF	4	63.18%
	DSR (ours)	PreAct-ResNet-18	IF	50	$\underline{67.74\%}$
DVS-CIFAR10	ASF-BP [39]	VGG-7	IF	50	62.50%
	Tandem Learning [40]	7-layer CNN	IF	20	65.59%
	STBP [42]	7-layer CNN	LIF	40	60.50%
	STBP-tdBN [48]	ResNet-19	LIF	10	67.80%
	Fang *et al.* [13]	7-layer CNN	LIF	20	74.80%
	DSR (ours)	VGG-11	IF	20	$\underline{75.03\% \pm 0.39\%}$
		VGG-11	LIF	20	$\mathbf{77.27\% \pm 0.24\%}$

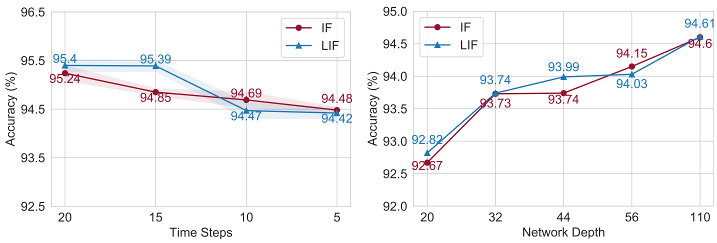
[1] Self-implemented results for ANN.

architectures. Although some direct training methods use smaller time steps than ours, our method can also achieve better performances than others when the number of time steps $N = 15, 10$, and 5, as shown in Fig. 3a.

For the ImageNet dataset, we also use the pre-activation ResNet-18 network architecture. To accelerate training, we adopt the hybrid training technique [31, 32]. And considering the data complexity and the 1000 classes, we use a moderate number of time steps to achieve satisfactory results. Our proposed method can outperform the direct training methods even if they use larger network architectures. Although some ANN-to-SNN methods have better accuracy, they use much more time steps than ours.

For the neuromorphic dataset DVS-CIFAR10, we adopt the VGG-11 architecture [36] and conduct 3 runs of experiments for each neural model. It can be found in Tab. 2 that the proposed DSR method outperforms other state-of-the-art methods with low latency using both the IF and the LIF models.

5.2. Model Validation and Ablation Study



(a) (b)

Figure 3. (a) Accuracies achieved by DSR method on CIFAR-10 with low lentency. The PreAct-ResNet-18 network architecture is used. Results are based on 3 runs of experiments. (b) Accuracies achieved by DSR method on CIFAR-10 with different network architectures. The number of time steps is 20. Narrow network architectures are used. See the Supplementary Materials for details about the architectures.

Effectiveness of the Proposed Method with Low Latency We validate that the proposed method can achieve competitive performances even with ultra-low latency, as shown in Fig. 3a. From 20 to 5 time steps, the IF and LIF models only suffer from less than 1% accuracy drop. The results for 5 time steps also outperform other state-of-the-art shown in Tab. 2. In the experiments, each model is trained from scratch, and we choose smaller learning rate for the total time steps $N = 5$. More training details can be found in the Supplementary Materials.

Effectiveness of the Proposed Method with Deep Network Structure Many SNN learning methods cannot adapt to deep network architectures, limiting the potential of SNNs. The reason is that the error for gradient approximation or ANN-to-SNN conversion will accumulate through layers, or the methods are computationally expensive for large-scale network structures. In this part, we test the proposed method on CIFAR-10 using pre-activation ResNet with different depths, namely 20, 32, 44, 56, 110 layers. Note that the channel size is smaller than the PreAct-ResNet-18, since the deep networks with large channel size as in PreAct-ResNet-18 perform not much better and are harder to train even for ANNs. More details about network architectures can be found in the Supplementary Materials. Results are shown in Fig. 3b. The figure shows that our method is effective on deep networks (>100 layers), and performs better with deeper network structures. This indicates the great potential of our method to achieve more advanced performance when using very deep networks.

Ablation Study on Methods to Reduce Representation Error We conduct the ablation study on the representation error reduction methods, namely training the threshold and introducing a new hyperparameter for the neural models. The models are trained on CIFAR-10 with PreAct-ResNet-18 structure and 20 time steps, and the results are shown in Tab. 3. The experiments imply that the representation error significantly hinders training and also demonstrate the superiority of the two methods to reduce the representation error. Note that the threshold training method also helps stabilize training, since the results become unstable for large thresholds without this method (*e.g.*, the standard deviation is 1.84% when $V_{th} = 6$). Furthermore, the average accuracy of not using both methods is better than the one of only using the firing mechanism modification, maybe due to the instability of the results when V_{th} is large.

Table 3. Ablation study on the representation error reduction methods on CIFAR-10. The PreAct-ResNet-18 architecture with the IF model is used, and results are based on 3 runs of experiments. 'F' means firing mechanism modification, and 'T' means threshold training.

Setting	Accuracy
DSR, init. $V_{th} = 6$	$95.24\% \pm 0.17\%$
DSR w/o F, init. $V_{th} = 6$	$92.88\% \pm 0.25\%$
DSR w/o T, $V_{th} = 6$	$90.45\% \pm 1.84\%$
DSR w/o T, $V_{th} = 2$	$90.47\% \pm 0.12\%$
DSR w/o F&T, $V_{th} = 6$	$92.59\% \pm 0.81\%$

6. Conclusion and Discussions

In this work, we show that the forward computation of SNNs can be represented as some sub-differentiable mapping. Based on the SNN-to-mapping representation, we propose the DSR method to train SNNs that avoids the non-differentiability problem in SNN training and does not require backpropagation through the temporal domain. We also analyze the representation error due to the small number of time steps, and propose to train the thresholds and introduce a new hyperparameter for the IF and LIF models to reduce the representation error. With the error reduction methods, we can train SNNs with low latency by the DSR method. Experiments show that the proposed method could achieve state-of-the-art performance on mainstream vision tasks, and show the effectiveness of the method when dealing with ultra-low latency or very deep network structures.

Societal impact and limitations. As for societal impact, there is no direct negative societal impact since this work only focuses on training SNNs. In fact, the development of high-performance low-latency SNNs allows SNNs to replace ANNs in some real-world tasks. This replacement will alleviate the huge energy consumption by ANNs and reduce carbon dioxide emissions. As for limitations, the DSR method may suffer from a certain degree of performance drop when the latency is extremely low (*e.g.*, with only 2 or 3 time steps), since the method requires relatively accurate spike representation to conduct backpropagation.

References

[1] 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 *NIPS*, 2018. 2

[2] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN*, 2000. 2

[3] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95(1):1–19, 2006. 3

[4] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *IJCV*, 113(1):54–66, 2015. 1

[5] Natalia Caporale and Yang Dan. Spike timing–dependent plasticity: a hebbian learning rule. *Annu. Rev. Neurosci.*, 31:25–46, 2008. 2

[6] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018. 6

[7] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. 1

[8] 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. 2

[9] Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *ICLR*, 2021. 1, 2, 3, 7

[10] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *IJCNN*, 2015. 2, 3

[11] Jianhao Ding, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. In *IJCAI*, 2021. 2, 3

[12] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *NIPS*, 2021. 1, 2, 3, 7

[13] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *ICCV*, 2021. 3, 7

[14] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *ECCV*, 2020. 2, 3

[15] 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 *CVPR*, 2020. 2, 3, 7

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 6

[17] Donald Olding Hebb. *The organisation of behaviour: a neuropsychological theory*. Science Editions New York, 1949. 2

[18] Yangfan Hu, Huajin Tang, Yueming Wang, and Gang Pan. Spiking deep residual network. *arXiv preprint arXiv:1805.01352*, 2018. 1

[19] Dongsung Huh and Terrence J. Sejnowski. Gradient descent for spiking neural networks. In *NIPS*, 2018. 2

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

[21] 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

[22] Sei Joon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection. In *AAAI*, 2020. 2, 3

[23] 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*, 14:119, 2020. 4

[24] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS computational biology*, 4(10):e1000180, 2008. 2

[25] Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *ICML*, 2021. 2

[26] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. 1

[27] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE transactions on neural networks and learning systems*, 29(7):3227–3235, 2017. 2, 3

[28] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. 1, 2, 3

[29] Stefano Panzeri and Simon R Schultz. A unified approach to the study of temporal, correlational, and rate coding. *Neural Computation*, 13(6):1311–1349, 2001. 2

[30] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019. 1

[31] 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. 7

[32] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *ICLR*, 2020. 7

[33] 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. 1, 2, 3, 4

[34] 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. 1, 2, 3, 7

[35] Sumit Bam Shrestha and Garrick Orchard. SLAYER: spike layer error reassignment in time. In *NIPS*, 2018. 1, 2, 3

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014. 7

[37] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, pages 1–9, 2021. 3

[38] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019. 1

[39] Hao Wu, Yueyi Zhang, Wenming Weng, Yongting Zhang, Zhiwei Xiong, Zheng-Jun Zha, Xiaoyan Sun, and Feng Wu. Training spiking neural networks with accumulated spiking flow. In *AAAI*, 2021. 2, 3, 4, 7

[40] Jibin Wu, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan. A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *TNNLS*, 2021. 2, 3, 7

[41] 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. 1, 2, 3, 7

[42] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *AAAI*, 2019. 2, 3, 7

[43] Timo C Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):1–17, 2021. 2, 3

[44] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In *NIPS*, 2021. 2, 3, 4, 7

[45] Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *AAAI*, 2021. 1, 2, 3, 7

[46] Yukun Yang, Wenrui Zhang, and Peng Li. Backpropagated neighborhood aggregation for accurate training of spiking neural networks. In *ICML*, 2021. 2

[47] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *NIPS*, 2020. 2, 7

[48] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *AAAI*, 2021. 1, 2, 3, 4, 7

[49] Shibo Zhou, Xiaohua Li, Ying Chen, Sanjeev T Chandrasekaran, and Arindam Sanyal. Temporal-coded deep spiking neural network with easy training and robust performance. In *AAAI*, 2021. 2, 3