Towards Memory- and Time-Efficient Backpropagation for Training Spiking Neural Networks

Qingyan Meng[1,2], Mingqing Xiao[3], Shen Yan[4], Yisen Wang[3,5], Zhouchen Lin[3,5,6,*], Zhi-Quan Luo[1,2]
[1]The Chinese University of Hong Kong, Shenzhen [2]Shenzhen Research Institute of Big Data
[3]Key Lab. of Machine Perception (MoE), School of Artificial Intelligence, Peking University
[4]Center for Data Science, Peking University [5]Institute for Artificial Intelligence, Peking University
[6]Peng Cheng Laboratory

qingyanmeng@link.cuhk.edu.cn, {mingqing_xiao, yanshen, yisen.wang, zlin}@pku.edu.cn, luozq@cuhk.edu.cn

Abstract

Spiking Neural Networks (SNNs) are promising energy-efficient models for neuromorphic computing. For training the non-differentiable SNN methods, the backpropagation through time (BPTT) with surrogate gradients (SG) method has achieved high performance. However, this method suffers from considerable memory cost and training time during training. In this paper, we propose the Spatial Learning Through Time (SLTT) method that can achieve high performance while greatly improving training efficiency compared with BPTT. First, we show that the backpropagation of SNNs through the temporal domain contributes just a little to the final calculated gradients. Thus, we propose to ignore the unimportant routes in the computational graph during backpropagation. The proposed method reduces the number of scalar multiplications and achieves a small memory occupation that is independent of the total time steps. Furthermore, we propose a variant of SLTT, called SLTT-K, that allows backpropagation only at K time steps, then the required number of scalar multiplications is further reduced and is independent of the total time steps. Experiments on both static and neuromorphic datasets demonstrate superior training efficiency and performance of our SLTT. In particular, our method achieves state-of-the-art accuracy on ImageNet, while the memory cost and training time are reduced by more than 70% and 50%, respectively, compared with BPTT. Our code is available at https://github.com/qymeng94/SLTT.

1. Introduction

Regarded as the third generation of neural network models [35], Spiking Neural Networks (SNNs) have recently attracted wide attention. SNNs imitate the neurodynamics of power-efficient biological networks, where neurons communicate through spike trains (*i.e.*, time series of spikes). A spiking neuron integrates input spike trains into its membrane potential. After the membrane potential exceeds a threshold, the neuron fires a spike and resets its potential [21]. The spiking neuron is active only when it experiences spikes, thus enabling event-based computation. This characteristic makes SNNs energy-efficient when implemented on neuromorphic chips [38, 11, 43]. As a comparison, the power consumption of deep Artificial Neural Networks (ANNs) is substantial.

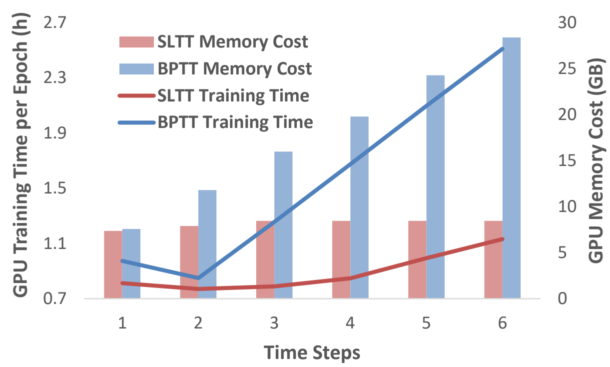

Figure 1: The training time and memory cost comparison between the proposed SLTT-1 method and the BPTT with SG method on ImageNet. SLTT-1 achieves similar accuracy as BPTT, while owning better training efficiency than BPTT both theoretically and experimentally. Please refer to Secs. 4 and 5 for details.

The computation of SNNs with discrete simulation can share a similar functional form as recurrent neural networks (RNNs) [40]. The unique component of SNNs is the non-differentiable threshold-triggered spike generation function. The non-differentiability, as a result, hinders the effective

*Corresponding author.

adoption of gradient-based optimization methods that can train RNNs successfully. Therefore, SNN training is still a challenging task. Among the existing SNN training methods, backpropagation through time (BPTT) with surrogate gradient (SG) [9, 49] has recently achieved high performance on complicated datasets in a small number of time steps (*i.e.*, short length of spike trains). The BPTT with SG method defines well-behaved surrogate gradients to approximate the derivative of the spike generation function. Thus the SNNs can be trained through the gradient-based BPTT framework [52], just like RNNs. With such framework, gradients are backpropagated through both the layer-by-layer spatial domain and the temporal domain. Accordingly, BPTT with SG suffers from considerable memory cost and training time that are proportional to the network size and the number of time steps. The training cost is further remarkable for large-scale datasets, such as ImageNet.

In this paper, we develop the Spatial Learning Through Time (SLTT) method that can achieve high performance while significantly reducing the training time and memory cost compared with the BPTT with SG method. We first decompose the gradients calculated by BPTT into spatial and temporal components. With the decomposition, the temporal dependency in error backpropagation is explicitly presented. We then analyze the contribution of temporal information to the final calculated gradients, and propose the SLTT method to delete the unimportant routes in the computational graph for backpropagation. In this way, the number of scalar multiplications is reduced; thus, the training time is reduced. SLTT further enables online training by calculating gradient instantaneously at each time step, without the requirement of storing information of other time steps. Then the memory occupation is independent of the number of total time steps, avoiding the significant training memory costs of BPTT. Due to the instantaneous gradient calculation, we also propose the SLTT-K method that conducts backpropagation only at K time steps. SLTT-K can further reduce the time complexity without performance loss. With the proposed techniques, we can obtain high-performance SNNs with superior training efficiency. The wall-clock training time and memory costs of SLTT-1 and BPTT on ImageNet under the same experimental settings are shown in Fig. 1. Formally, our contributions include:

1. Based on our analysis of error backpropagation in SNNs, we propose the Spatial Learning Through Time (SLTT) method to achieve better time and memory efficiency than the commonly used BPTT with SG method. Compared with the BPTT with SG method, the number of scalar multiplications is reduced, and the training memory is constant with the number of time steps, rather than grows linearly with it.

2. Benefiting from our online training framework, we propose the SLTT-K method that further reduces the time complexity of SLTT. The required number of scalar multiplication operations is reduced from $\Omega(T)$[1] to $\Omega(K)$, where T is the number of total time steps, and $K < T$ is the parameter indicating the number of time steps to conduct backpropagation.

3. Our models achieve competitive SNN performance with superior training efficiency on CIFAR-10, CIFAR-100, ImageNet, DVS-Gesture, and DVS-CIFAR10 under different network settings or large-scale network structures. On ImageNet, our method achieves state-of-the-art accuracy while the memory cost and training time are reduced by more than 70% and 50%, respectively, compared with BPTT.

2. Related Work

The BPTT Framework for Training SNNs. A natural methodology for training SNNs is to adopt the gradient-descent-based BPTT framework, while assigning surrogate gradients (SG) to the non-differentiable spike generation functions to enable meaningful gradient calculation [66, 40, 56, 57, 50, 26, 34]. Under the BPTT with SG framework, many effective techniques have been proposed to improve the performance, such as threshold-dependent batch normalization [67], carefully designed surrogate functions [32] or loss functions [23, 15], SNN-specific network structures [19], and trainable parameters of neuron models [20]. Many works conduct multi-stage training, typically including an ANN pre-training process, to reduce the latency (*i.e.*, the number of time steps) for the energy efficiency issue, while maintaining competitive performance [47, 46, 8, 7]. The BPTT with SG method has achieved high performance with low latency on both static [19, 22] and neuromorphic [33, 15] datasets. However, those approaches need to backpropagate error signals through both temporal and spatial domains, thus suffering from high computational costs during training [13]. In this work, we reduce the memory and time complexity of the BPTT with SG framework with gradient approximation and instantaneous gradient calculation, while maintaining the same level of performance.

Other SNN Training Methods. The ANN-to-SNN conversion method [61, 14, 49, 48, 25, 24, 16] has recently yielded top performance, especially on ImageNet [31, 37, 5]. This method builds a connection between the firing rates of SNNs and some corresponding ANN outputs. With this connection, the parameters of an SNN are directly determined from the associated ANN. Despite the good performance, the required latency is much higher compared with

[1] $f(x) = \Omega(g(x))$ means that there exist $c > 0$ and $n > 0$, such that $0 \leq cg(x) \leq f(x)$ for all $x \geq n$.

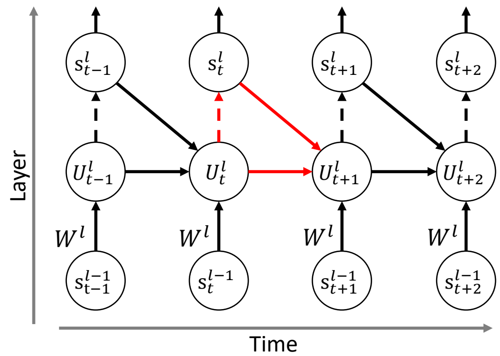

Figure 2: Computational graph of multi-layer SNNs. Dashed arrows represent the non-differentiable spike generation functions.

the BPTT with SG method. This fact hurts the energy efficiency of SNN inference [10]. Furthermore, the conversion method is not suitable for neuromorphic data. Some gradient-based direct training methods find the equivalence between spike representations (*e.g.*, firing rates or first spike times) of SNNs and some differentiable mappings or fixed-point equations [39, 68, 59, 36, 51, 54, 55, 60, 62]. Then the spike-representation-based methods train SNNs by gradients calculated from the corresponding mappings or fixed-point equations. Such methods have recently achieved competitive performance, but still suffer relatively high latency, like the conversion-based methods. To achieve low latency, our work is mainly based on the BPTT with SG method and then focuses on the training cost issue of BPTT with SG.

Efficient Training for SNNs. Several RNN training methods pursue online learning and constant memory occupation agnostic time horizon, such as real time recurrent learning [53] and forward propagation through time [27]. Inspired by them, some SNN training methods [64, 65, 2, 3, 63] apply similar ideas to achieve memory-efficient and online learning. However, such SNN methods cannot scale to large-scale tasks due to some limitations, such as using feedback alignment [41], simple network structures, and still large memory costs although constant with time. [28] ignores temporal dependencies of information propagation to enable local training with no memory overhead for computing gradients. They use similar ways as ours to approximate the gradient calculation, but do not verify the reasonableness of the approximation, and cannot achieve comparable accuracy as ours, even for simple tasks. [44] presents the sparse SNN backpropagation algorithm in which gradients only backpropagate through "active neurons", that account for a small number of the total, at each time step. However, [44] does not consider large-scale tasks, and the memory grows linearly with the number of time steps. Recently, some methods [62, 58] have achieved satisfactory performance on large-scale datasets with time steps-independent memory occupation. Still, they either rely on pre-trained ANNs and cannot conduct direct training[62], or do not consider reducing time complexity and require more memory than our work due to tracking presynaptic activities [58]. Our work can achieve state-of-the-art (SOTA) performance while maintaining superior time and memory efficiency compared with other methods.

3. Preliminaries

3.1. The Leaky Integrate and Fire Model

A spiking neuron replicates the behavior of a biological neuron which integrates input spikes into its membrane potential $u(t)$ and transmits spikes when the potential u reaches a threshold. Such spike transmission is controlled via some spiking neural models. In this paper, we consider a widely adopted neuron model, the leaky integrate and fire (LIF) model [6], to characterize the dynamics of $u(t)$:

$$\tau \frac{\mathrm{d}u(t)}{\mathrm{d}t} = -(u(t) - u_{rest}) + R \cdot I(t), \text{ when } u(t) < V_{th}, \tag{1}$$

where τ is the time constant, R is the resistance, u_{rest} is the resting potential, V_{th} is the spike threshold, and I is the input current which depends on received spikes. The current model is given as $I(t) = \sum_i w_i' s_i(t) + b'$, where w_i' is the weight from neuron-i to the target neuron, b' is a bias term, and $s_i(t)$ is the received train from neuron-i. $s_i(t)$ is formed as $s_i(t) = \sum_f \delta(t - t_{i,f})$, in which $\delta(\cdot)$ is the Dirac delta function and $t_{i,f}$ is the f-th fire time of neuron-i. Once $u \geq V_{th}$ at time t_f, the neuron output a spike, and the potential is reset to u_{rest}. The output spike train is described as $s_{out}(t) = \sum_f \delta(t - t_f)$.

In application, the discrete computational form of the LIF model is adopted. With $u_{rest} = 0$, the discrete LIF model can be described as

$$\begin{cases} u[t] = (1 - \frac{1}{\tau})v[t-1] + \sum_i w_i s_i[t] + b, \\ s_{out}[t] = H(u[t] - V_{th}), \\ v[t] = u[t] - V_{th} s_{out}[t], \end{cases} \tag{2}$$

where $t \in \{1, 2, \cdots, T\}$ is the time step index, $H(\cdot)$ is the Heaviside step function, $s_{out}[t], s_i[t] \in \{0, 1\}$, $v[t]$ is the intermediate value representing the membrane potential before being reset and $v[0] = 0$, and w_i and b are reparameterized version of w_i' and b', respectively, where τ and R are absorbed. The discrete step size is 1, so $\tau > 1$ is required.

3.2. Backpropagation Through Time with Surrogate Gradient

Consider the multi-layer feedforward SNNs with the LIF neurons based on Eq. (2):

$$\mathbf{u}^l[t] = (1 - \frac{1}{\tau})(\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]) + \mathbf{W}^l \mathbf{s}^{l-1}[t], \tag{3}$$

where $l = 1, 2, \cdots, L$ is the layer index, $t = 1, 2, \cdots, T$, $0 < 1 - \frac{1}{\tau} < 1$, $\mathbf{s}^0$ are the input data to the network, $\mathbf{s}^l$ are the output spike trains of the l^{th} layer, $\mathbf{W}^l$ are the weight to be trained. We ignore the bias term for simplicity. The final output of the network is $\mathbf{o}[t] = \mathbf{W}^o \mathbf{s}^L[t]$, where $\mathbf{W}^o$ is the parameter of the classifier. The classification is based on the average of the output at each time step $\frac{1}{T}\sum_{t=1}^{T} \mathbf{o}[t]$. The loss function $\mathcal{L}$ is defined on $\{\mathbf{o}[1], \cdots, \mathbf{o}[T]\}$, and is often defined as [67, 46, 59, 32]

$$\mathcal{L} = \ell(\frac{1}{T}\sum_{t=1}^{T} \mathbf{o}[t], y), \tag{4}$$

where y is the label, and ℓ can be the cross-entropy function.

BPTT with SG calculates gradients according to the computational graph of Eq. (3) shown in Fig. 2. The pseudocode is described in the Supplementary Materials. For each neuron i in the l-th layer, the derivative $\frac{\partial \mathbf{s}_i^l[t]}{\partial \mathbf{u}_i^l[t]}$ is zero for all values of $\mathbf{u}_i^l[t]$ except when $\mathbf{u}_i^l[t] = V_{th}$, where the derivative is infinity. Such a non-differentiability problem is solved by approximating $\frac{\partial \mathbf{s}_i^l[t]}{\partial \mathbf{u}_i^l[t]}$ with some well-behaved surrogate function, such as the rectangle function [56, 57]

$$\frac{\partial s}{\partial u} = \frac{1}{\gamma}\mathbb{1}\left(|u - V_{th}| < \frac{\gamma}{2}\right), \tag{5}$$

and the triangle function [15, 17]

$$\frac{\partial s}{\partial u} = \frac{1}{\gamma^2}\max\left(0, \gamma - |u - V_{th}|\right), \tag{6}$$

where $\mathbb{1}(\cdot)$ is the indicator function, and the hyperparameter γ for both functions is often set as V_{th}.

4. The proposed Spatial Learning Through Time Method

4.1. Observation from the BPTT with SG Method

In this subsection, we decompose the derivatives for membrane potential, as calculated in the BPTT method, into spatial components and temporal components. Based on the decomposition, we observe that the spatial components dominate the calculated derivatives. This phenomenon inspires the proposed method, as introduced in Sec. 4.2.

According to Eq. (3) and Fig. 2, the gradients for weights in an SNN with T time steps are calculated by

$$\nabla_{\mathbf{W}^l}\mathcal{L} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]}^\top \mathbf{s}^{l-1}[t]^\top,\ l = L, L-1, \cdots, 1. \tag{7}$$

We further define

$$\epsilon^l[t] \triangleq \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \tag{8}$$

as the sensitivity of $\mathbf{u}^l[t+1]$ with respect to $\mathbf{u}^l[t]$, represented by the red arrows shown in Fig. 2. Then with the chain rule, $\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]}$ in Eq. (7) can be further calculated recursively. In particular, for the output layer, we arrive at

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^L[t]} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t]}\frac{\partial \mathbf{s}^L[t]}{\partial \mathbf{u}^L[t]} + \sum_{t'=t+1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t']}\frac{\partial \mathbf{s}^L[t']}{\partial \mathbf{u}^L[t']}\prod_{t''=1}^{t'-t}\epsilon^L[t'-t''], \tag{9}$$

and for the intermediate layer $l = L-1, \cdots, 1$, we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]} =& \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]}\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \\ &+ \sum_{t'=t+1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t']}\frac{\partial \mathbf{u}^{l+1}[t']}{\partial \mathbf{s}^l[t']}\frac{\partial \mathbf{s}^l[t']}{\partial \mathbf{u}^l[t']}\prod_{t''=1}^{t'-t}\epsilon^l[t'-t''].\end{aligned} \tag{10}$$

The detailed derivation can be found in the Supplementary Materials. In both Eqs. (9) and (10), the terms before the addition symbols on the R.H.S. (the blue terms) can be treated as the spatial components, and the remaining parts (the green terms) represent the temporal components.

We observe that the temporal components contribute a little to $\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]}$, since the diagonal matrix $\prod_{t''=1}^{t'-t}\epsilon^l[t'-t'']$ is supposed to have a small spectral norm for typical settings of surrogate functions. To see this, we consider the rectangle surrogate (Eq. (5)) with $\gamma = V_{th}$ as an example. Based on Eq. (3), the diagonal elements of $\epsilon^l[t]$ are

$$\left(\epsilon^l[t]\right)_{jj} = \begin{cases} 0, & \frac{1}{2}V_{th} < \left(\mathbf{u}^l[t]\right)_j < \frac{3}{2}V_{th}, \\ 1 - \frac{1}{\tau}, & \text{otherwise}. \end{cases} \tag{11}$$

Define $\lambda \triangleq 1 - \frac{1}{\tau}$, then $\left(\epsilon^l[t]\right)_{jj}$ is zero in an easily-reached interval, and is at least not large for commonly used small λ (*e.g.*, $\lambda = 0.5$ [58, 15], $\lambda = 0.25$ [67], and $\lambda = 0.2$ [23]). The diagonal values of the matrix $\prod_{t''=1}^{t'-t}\epsilon^l[t'-t'']$ are smaller than the single term $\epsilon^l[t'-t'']$ due to the product operations, especially when $t'-t$ is large. The temporal components are further unimportant if the spatial and temporal components have similar directions. Then the spatial components in Eqs. (9) and (10) dominate the gradients.

For other widely-used surrogate functions and their corresponding hyperparameters, the phenomenon of dominant spatial components still exists since the surrogate functions have similar shapes and behavior. In order to illustrate this, we conduct experiments on CIFAR-10, DVS-CIFAR10, and ImageNet using the triangle surrogate (Eq. (6)) with $\gamma = V_{th}$. We use the BPTT with SG method to train the SNNs on the abovementioned three datasets, and call the calculated gradients the baseline gradients. During training, we also calculate the gradients for weights when the temporal components are abandoned, and call such gradients the spatial gradients. We compare the disparity between baseline and spatial gradients by calculating their cosine similarity. The results are demonstrated in Fig. 3. The similarity maintains

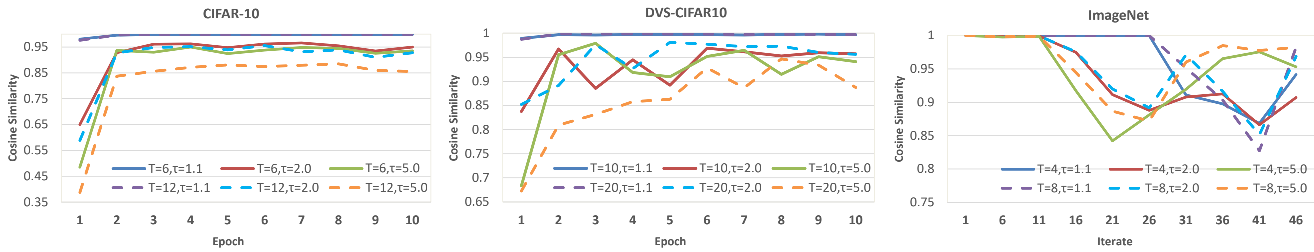

Figure 3: The cosine similarity between the gradients calculated by BPTT and the "spatial gradients". For the CIFAR-10, DVS-CIFAR10, and ImageNet datasets, the network architectures of ResNet-18, VGG-11, and ResNet-34 are adopted, respectively. Other settings and hyperparameters for the experiments are described in the Supplementary Materials. We calculate the cosine similarity for different layers and report the average in the figure. For ImageNet, we only train the network for 50 iterates since the training is time-consuming. Dashed curves represent a larger number of time steps.

a high level for different datasets, the number of time steps, and τ. In particular, for $\tau = 1.1$ ($\lambda = 1 - \frac{1}{\tau} \approx 0.09$), the baseline and spatial gradients consistently have a remarkably similar direction on CIFAR-10 and DVS-CIFAR10. In conclusion, the spatial components play a dominant role in the gradient backpropagation process.

4.2. Spatial Learning Through Time

Based on the observation introduced in Sec. 4.1, we propose to ignore the temporal components in Eqs. (9) and (10) to achieve more efficient backpropagation. In detail, the gradients for weights are calculated by

$$\nabla_{\mathbf{W}^l}\mathcal{L} = \sum_{t=1}^{T} \mathbf{e}_{\mathbf{W}}^l[t], \quad \mathbf{e}_{\mathbf{W}}^l[t] = \mathbf{e}_{\mathbf{u}}^l[t]^\top \mathbf{s}^{l-1}[t]^\top, \tag{12}$$

where

$$\mathbf{e}_{\mathbf{u}}^l[t] = \begin{cases} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t]} \frac{\partial \mathbf{s}^L[t]}{\partial \mathbf{u}^L[t]}, & l = L, \\ \mathbf{e}_{\mathbf{u}}^{l+1}[t] \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}, & l < L, \end{cases} \tag{13}$$

and $\mathbf{e}_{\mathbf{u}}^l[t]$ is a row vector. Compared with Eqs. (7), (9) and (10), the required number of scalar multiplications in Eqs. (12) and (13) is reduced from $\Omega(T^2)$ to $\Omega(T)$. Note that the BPTT method does not conduct naive computation of the sum-product as shown in Eqs. (9) and (10), but in a recursive way to achieve $\Omega(T)$ computational complexity, as shown in the Supplementary Materials. Although BPTT and the proposed update rule both need $\Omega(T)$ scalar multiplications, such multiplication operations are reduced due to ignoring some routes in the computational graph. Please refer to Supplementary Materials for time complexity analysis. Therefore, the time complexity of the proposed update rule is much lower than that of BPTT with SG, although they are both proportional to T.

According to Eqs. (12) and (13), the error signals $\mathbf{e}_{\mathbf{W}}^l$ and $\mathbf{e}_{\mathbf{u}}^l$ at each time step can be calculated independently without information from other time steps. Thus, if $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t]}$ can be calculated instantaneously at time step t, $\mathbf{e}_{\mathbf{W}}^l[t]$ and $\mathbf{e}_{\mathbf{u}}^l[t]$ can also be calculated instantaneously at time step t. Then there is no need to store intermediate states of the whole time horizon. To achieve the instantaneous calculation of $\frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t]}$, we adopt the loss function [23, 15, 58]

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \ell(\mathbf{o}[t], y), \tag{14}$$

which is an upper bound of the loss introduced in Eq. (4).

Algorithm 1 One iteration of SNN training with the SLTT or SLTT-K methods.

Input: Time steps T; Network depth L; Network parameters $\{\mathbf{W}^l\}_{l=1}^L$; Training data $(\mathbf{s}^0, \mathbf{y})$; Learning rate η; Required backpropagation times K (for SLTT-K).
Initialize: $\Delta \mathbf{W}^l = 0,\ l = 1, 2, \cdots, L$.

```
if using SLTT-K then
    Sample K numbers in [1, 2, ···, T] w/o replacement to form required_bp_steps;
else
    required_bp_steps = [1, 2, ···, T];
end if
for t = 1, 2, ···, T do
    Calculate s^L[t] by Eqs. (2) and (3);      //Forward
    Calculate the instantaneous loss ℓ in Eq. (14);
    if t in required_bp_steps then      //Backward
        e_u^L[t] = (1/T) (∂ℓ/∂s^L[t]) (∂s^L[t]/∂u^L[t]);
        for l = L − 1, ···, 1 do
            e_u^l[t] = e_u^{l+1}[t] (∂u^{l+1}[t]/∂s^l[t]) (∂s^l[t]/∂u^l[t]);
            ΔW^l += e_u^l[t]^⊤ s^{l−1}[t]^⊤;
        end for
    end if
end for
W^l = W^l − η ΔW^l, l = 1, 2, ···, L;
```

Output: Trained network parameters $\{\mathbf{W}^l\}_{l=1}^L$.

We propose the Spatial Learning Through Time (SLTT) method using gradient approximation and instantaneous gradient calculation, as detailed in Algorithm 1. In Algorithm 1, all the intermediate terms at time step t, such as $\mathbf{e}_{\mathbf{u}}^{l}[t], \mathbf{s}^{l}[t], \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^{l}[t]}$, and $\frac{\partial \mathbf{s}^{l}[t]}{\partial \mathbf{u}^{l}[t]}$, are never used in other time steps, so the required memory overhead of SLTT is constant agnostic to the total number of time steps T. On the contrary, the BPTT with SG method has an $\Omega(T)$ memory cost associated with storing all intermediate states for all time steps. In summary, the proposed method is both time-efficient and memory-efficient, and has the potential to enable online learning for neuromorphic substrates [65].

4.3. Further Reducing Time Complexity

Due to the online update rule of the proposed method, the gradients for weights are calculated according to an ensemble of T independent computational graphs, and the time complexity of gradient calculation is $\Omega(T)$. The T computational graphs can have similar behavior, and then similar gradient directions can be obtained with only a portion of the computational graphs. Based on this, we propose to train a portion of time steps to reduce the time complexity further. In detail, for each iteration in the training process, we randomly choose K time indexes from the time horizon, and only conduct backpropagation with SLTT at the chosen K time steps. We call such a method the SLTT-K method, and the pseudo-code is given in Algorithm 1. Note that setting $K = T$ results in the original SLTT method. Compared with SLTT, the time complexity of SLTT-K is reduced to $\Omega(K)$, and the memory complexity is the same. In our experiments, SLTT-K can achieve satisfactory performance even when $K = 1$ or 2, as shown in Sec. 5, indicating superior efficiency of the SLTT-K method.

5. Experiments

In this section, we evaluate the proposed method on CIFAR-10 [29], CIFAR-100[29], ImageNet[12], DVS-Gesture[1], and DVS-CIFAR10 [30] to demonstrate its superior performance regarding training costs and accuracy. For our SNN models, we set $V_{th} = 1$ and $\tau = 1.1$, and apply the triangle surrogate function (Eq. (6)). An effective technique, batch normalization (BN) along the temporal dimension [67], cannot be adopted to our method, since it requires calculation along the total time steps and then intrinsically prevents time-steps-independent memory costs. Therefore, for some tasks, we borrow the idea from normalization-free ResNets (NF-ResNets) [4] to replace BN by weight standardization (WS) [45]. Please refer to the Supplementary Materials for experimental details.

Table 1: Comparison of training memory cost, training time, and accuracy between SLTT and BPTT. The "Memory" column indicates the maximum memory usage on an GPU during training. And the "Time" column indicates the wall-clock training time.

Dataset	Method	Memory	Time	Acc
CIFAR-10	BPTT	3.00G	6.35h	**94.60%**
	SLTT	**1.09G**	**4.58h**	94.59%
CIFAR-100	BPTT	3.00G	6.39h	73.80%
	SLTT	**1.12G**	**4.68h**	**74.67%**
ImageNet	BPTT	28.41G	73.8h	**66.47%**
	SLTT	**8.47G**	**66.9h**	66.19%
DVS-Gesture	BPTT	5.82G	2.68h	97.22%
	SLTT	**1.07G**	**2.64h**	**97.92%**
DVS-CIFAR10	BPTT	3.70G	4.47h	73.60%
	SLTT	**1.07G**	**3.43h**	**77.30%**

5.1. Comparison with BPTT

The major advantage of SLTT over BPTT is the low memory and time complexity. To verify the advantage of SLTT, we use both methods with the same experimental setup to train SNNs. For CIFAR-10, CIFAR-100, ImageNet, DVS-Gesture, and DVS-CIFAR10, the network architectures we adopt are ResNet-18, ResNet-18, NF-ResNet-34, VGG-11, and VGG-11, respectively, and the total number of time steps are 6, 6, 6, 20, and 10, respectively. For ImageNet, to accelerate training, we first train the SNN with only 1 time step for 100 epochs to get a pre-trained model, and then use SLTT or BPTT to fine-tune the model with 6 time steps for 30 epochs. Details of the training settings can be found in the Supplementary Materials. We run all the experiments on the same Tesla-V100 GPU, and ensure that the GPU card is running only one experiment at a time to perform a fair comparison. It is not easy to directly compare the running time for two training methods since the running time is code-dependent and platform-dependent. In our experiments, we measure the wall-clock time of the total training process, including forward propagation and evaluation on the validation set after each epoch, to give a rough comparison. For ImageNet, the training time only includes the 30-epoch fine-tuning part.

The results of maximum memory usage, total wall-clock training time, and accuracy for both SLTT and BPTT on different datasets are listed in Tab. 1. SLTT enjoys similar accuracy compared with BPTT while using less memory and time. For all the datasets, SLTT requires less than one-third of the GPU memory of BPTT. In fact, SLTT maintains constant memory cost over the different number of time steps T, while the training memory of BPTT grows linearly in T. The memory occupied by SLTT for T time steps is always similar to that of BPTT for 1 time step. Regarding training time, SLTT also enjoys faster training on both algorithmic

Table 2: Comparison of training time and accuracy between SLTT and SLTT-K. "NFRN" means Normalizer-Free ResNet. For DVS-Gesture and DVS-CIFAR10, the "Acc" column reports the average accuracy of 3 runs of experiments using different random seeds. We skip the standard deviation values since they are almost 0, except for SLTT on DVS-CIFAR10 where the value is 0.23%.

Network	Method	Memory	Time	Acc
DVS-Gesture, $T = 20$				
VGG-11	SLTT	≈1.1G	2.64h	**97.92%**
	SLTT-4		**1.69h**	97.45%
DVS-CIFAR10, $T = 10$				
VGG-11	SLTT	≈1.1G	3.43h	**77.16%**
	SLTT-2		**2.49h**	76.70%
ImageNet, $T = 6$				
NFRN-34	SLTT	≈8.5G	66.90h	**66.19%**
	SLTT-2		41.88h	66.09%
	SLTT-1		**32.03h**	66.17%
NFRN-50	SLTT	≈24.5G	126.05h	**67.02%**
	SLTT-2		80.63h	66.98%
	SLTT-1		**69.36h**	66.94%
NFRN-101	SLTT	≈33.8G	248.23h	69.14%
	SLTT-2		123.05h	**69.26%**
	SLTT-1		**91.73h**	69.14%

and practical aspects. For DVS-Gesture, the training time for both methods are almost the same, deviating from the algorithmic time complexity. That may be due to really little training time for both methods and the good parallel computing performance of the GPU.

5.2. Performance of SLTT-K

As introduced in Sec. 4.3, the proposed SLTT method has a variant, SLTT-K, that conducts backpropagation only in randomly selected K time steps for reducing training time. We verify the effectiveness of SLTT-K on the neuromorphic datasets, DVS-Gesture and DVS-CIFAR10, and the large-scale static dataset, ImageNet. For the ImageNet dataset, we first pre-train the 1-time-step networks, and then fine-tune them with 6 time steps, as described in Sec. 5.1. We train the NF-ResNet-101 networks on a single Tesla-A100 GPU, while we use a single Tesla-V100 GPU for other experiments. As shown in Tab. 2, the SLTT-K method yields competitive accuracy with SLTT (also BPTT) for different datasets and network architectures, even when $K = \frac{1}{6}T$ or $\frac{1}{5}T$. With such small values of K, further compared with BPTT, the SLTT-K method enjoys comparable or even better training results, less memory cost (much less if T is large), and much faster training speed.

Table 3: Comparison of training memory cost and training time per epoch between SLTT and OTTT.

Dataset	Method	Memory	Time/Epoch
CIFAR-10	OTTT	1.71G	184.68s
	SLTT	**1.00G**	**54.48s**
CIFAR-100	OTTT	1.71G	177.72s
	SLTT	**1.00G**	**54.60s**
ImageNet	OTTT	19.38G	7.52h
	SLTT	**8.47G**	**2.23h**
DVS-Gesture	OTTT	3.38G	236.64s
	SLTT	**2.08G**	**67.20s**
DVS-CIFAR10	OTTT	4.32G	114.84s
	SLTT	**1.90G**	**48.00s**

5.3. Comparison with Other Efficient Training Methods

There are other online learning methods for SNNs [58, 2, 3, 63, 62] that achieve time-steps-independent memory costs. Among them, OTTT [58] enables direct training on large-scale datasets with relatively low training costs. In this subsection, we compare SLTT and OTTT under the same experimental settings of network structures and total time steps (see Supplementary Materials for details). The wall-clock training time and memory cost are calculated based on 3 epochs of training. The two methods are comparable since the implementation of them are both based on PyTorch [42] and SpikingJelly [18]. The results are shown in Tab. 3. SLTT outperforms OTTT on all the datasets regarding memory costs and training time, indicating the superior efficiency of SLTT. As for accuracy, SLTT also achieves better results than OTTT, as shown in Tab. 4.

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

The proposed SLTT method is not designed to achieve the best accuracy, but to enable more efficient training. Still, our method achieves competitive results compared with the SOTA methods, as shown in Tab. 4. Besides, our method obtains such good performance with only a few time steps, leading to low energy consumption when the trained networks are implemented on neuromorphic hardware.

For the BPTT-based methods, there is hardly any implementation of large-scale network architectures on ImageNet due to the significant training costs. To our knowledge, only Fang *et al.* [19] leverage BPTT to train an SNN with more than 100 layers, while the training process requires near 90G GPU memory for $T = 4$. Our SLTT-2 method succeeds in training the same-scale ResNet-101 network with only 34G memory occupation and 4.10h of training time per epoch (Tabs. 2 and 4). Compared with BPTT, the training memory and time of SLTT-2 are reduced by more than 70% and 50%, respectively. Furthermore, since the fo-

Table 4: Comparisons with other SNN training methods on CIFAR-10, CIFAR-100, ImageNet, DVS-Gesture, and DVS-CIFAR10. Results of our method on all the datasets, except ImageNet, are based on 3 runs of experiments. The "Efficient Training" column means whether the method requires less training time or memory occupation than the vanilla BPTT method for one epoch of training.

	Method	Network	Time Steps	Efficient Training	Mean±Std (Best)
CIFAR-10	LTL-Online[62] [1]	ResNet-20	16	✓	93.15%
	OTTT[58]	VGG-11 (WS)	6	✓	$93.52 \pm 0.06\%$ (93.58%)
	Dspike[32]	ResNet-18	6	✗	$94.25 \pm 0.07\%$
	TET[15]	ResNet-19	6	✗	$\mathbf{94.50} \pm \mathbf{0.07}\%$
	SLTT (ours)	ResNet-18	6	✓	$94.44\% \pm 0.21\%$ ($\mathbf{94.59}\%$)
CIFAR-100	OTTT[58]	VGG-11 (WS)	6	✓	$71.05 \pm 0.04\%$ (71.11%)
	ANN-to-SNN[5] [1]	VGG-16	8	✓	73.96%
	RecDis[23]	ResNet-19	4	✗	$74.10 \pm 0.13\%$
	TET[15]	ResNet-19	6	✗	$\mathbf{74.72} \pm \mathbf{0.28}\%$
	SLTT (ours)	ResNet-18	6	✓	$74.38\% \pm 0.30\%$ (74.67%)
ImageNet	ANN-to-SNN[31] [1]	ResNet-34	32	✓	64.54%
	TET[15]	ResNet-34	6	✗	64.79%
	OTTT[58]	NF-ResNet-34	6	✓	65.15%
	SEW [19]	Sew ResNet-34,50,101	4	✗	$67.04\%, 67.78\%, 68.76\%$
	SLTT (ours)	NF-ResNet-34,50	6	✓	$66.19\%, 67.02\%$
	SLTT-2 (ours)	NF-ResNet-101	6	✓	$\mathbf{69.26}\%$
DVS-Gesture	STBP-tdBN [67]	ResNet-17	40	✗	96.87%
	OTTT [58]	VGG-11 (WS)	20	✓	96.88%
	PLIF [20]	VGG-like	20	✗	97.57%
	SEW[19]	Sew ResNet	16	✗	97.92%
	SLTT (ours)	VGG-11	20	✓	$97.92 \pm 0.00\%$ (97.92%)
		VGG-11 (WS)	20	✓	$\mathbf{98.50} \pm \mathbf{0.21}\%$ ($\mathbf{98.62}\%$)
DVS-CIFAR10	Dspike[32]	ResNet-18	10	✗	$75.40 \pm 0.05\%$
	InfLoR[22]	ResNet-19	10	✗	$75.50 \pm 0.12\%$
	OTTT [58]	VGG-11 (WS)	10	✓	$76.27 \pm 0.05\%$(76.30%)
	TET[15]	VGG-11	10	✗	$\mathbf{83.17} \pm \mathbf{0.15}\%$
	SLTT (ours) [2]	VGG-11	10	✓	$77.17 \pm 0.23\%$ (77.30%)
	SLTT (ours)	VGG-11	10	✓	$82.20 \pm 0.95\%$ (83.10%)

[1] Pre-trained ANN models are required. [2] Without data augmentation.

cus of the SOTA BPTT-type methods (*e.g.*, surrogate function, network architecture, and regularization) are orthogonal to ours, our training techniques can be plugged into their methods to achieve better training efficiency. Some ANN-to-SNN-based and spike representation-based methods [62, 5, 31] also achieve satisfactory accuracy with relatively small training costs. However, they typically require a (much) larger number of time steps (Tab. 4), which hurts the energy efficiency for neuromorphic computing.

5.5. Influence of T and τ

For efficient training, the SLTT method approximates the gradient calculated by BPTT by ignoring the temporal components in Eqs. (9) and (10). So when T or τ is large, the approximation may not be accurate enough. In this subsection, we conduct experiments with different τ and T on the neuromorphic datasets, DVS-Gesture and DVS-CIFAR10. We verify that the proposed method can still work well for large T and commonly used τ [58, 15, 67, 23], as shown in Fig. 4. Regarding large time steps, SLTT obtains sim-

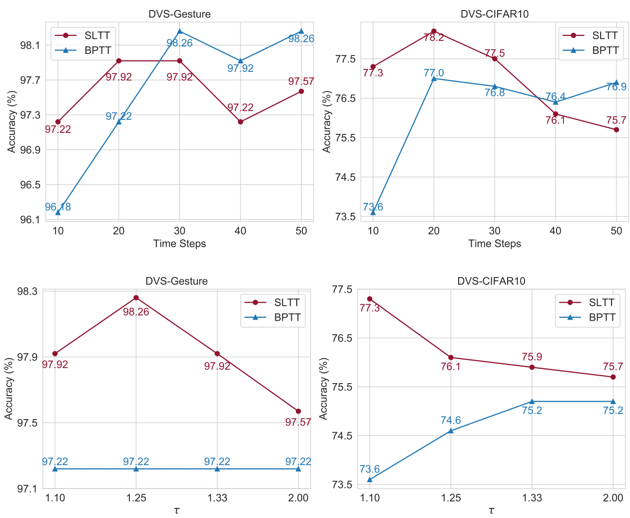

Figure 4: Performance of SLTT and BPTT for different number of time steps (the top two subfigures) and for different τ (the bottom two subfigures). Experiments are conducted on the neuromorphic datasets, DVS-Gesture and DVS-CIFAR10.

ilar accuracy with BPTT even when $T = 50$, and SLTT can outperform BPTT when $T < 30$ on the two neuromorphic datasets. For different τ, our method can consistently perform better than BPTT, although there is a performance drop for SLTT when τ is large.

6. Conclusion

In this work, we propose the Spatial Learning Through Time (SLTT) method that significantly reduces the time and memory complexity compared with the vanilla BPTT with SG method. We first show that the backpropagation of SNNs through the temporal domain contributes a little to the final calculated gradients. By ignoring unimportant temporal components in gradient calculation and introducing an online calculation scheme, our method reduces the scalar multiplication operations and achieves time-step-independent memory occupation. Additionally, thanks to the instantaneous gradient calculation in our method, we propose a variant of SLTT, called SLTT-K, that allows backpropagation only at K time steps. SLTT-K can further reduce the time complexity of SLTT significantly. Extensive experiments on large-scale static and neuromorphic datasets demonstrate superior training efficiency and high performance of the proposed method, and illustrate the method's effectiveness under different network settings and large-scale network structures.

Acknowledgment

The work of Z.-Q. Luo was supported by the National Natural Science Foundation of China under Grant 61731018, and the Guangdong Provincial Key Laboratory of Big Data Computation Theories and Methods. Z. Lin was supported by the NSF China (No. 61731018), NSFC Tianyuan Fund for Mathematics (No. 12026606), Project 2020BD006 supported by PKU-Baidu Fund, and Qualcomm. The work of Yisen Wang was partially supported by the National Natural Science Foundation of China under Grant 62006153, and Project 2020BD006 supported by PKU-Baidu Fund.

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 *CVPR*, 2017. 6

[2] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):1–15, 2020. 3, 7

[3] Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *TNNLS*, 2022. 3, 7

[4] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR, 2021. 6

[5] Tong Bu, Wei Fang, Jianhao Ding, Penglin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *ICLR*, 2022. 2, 8

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

[7] Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. One timestep is all you need: Training spiking neural networks with ultra low latency. *arXiv preprint arXiv:2110.05929*, 2021. 2

[8] Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning. In *ECCV*, 2022. 2

[9] Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. Surrogate gradients for analog neuromorphic computing. *Proc. Natl. Acad. Sci. USA*, 119(4):e2109194119, 2022. 2

[10] Simon Davidson and Steve B Furber. Comparison of artificial and spiking neural networks on digital hardware. *Frontiers in Neuroscience*, 15:651141, 2021. 3

[11] 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

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[13] 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

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

[15] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *ICLR*, 2022. 2, 4, 5, 8

[16] 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

[17] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. USA*, 113(41):11441–11446, 2016. 4

[18] Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. Spikingjelly. `https://github.com/fangwei123456/spikingjelly`, 2020. 7

[19] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *NeurIPS*, 2021. 2, 7, 8

[20] 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. 2, 8

[21] 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

[22] Yufei Guo, Yuanpei Chen, Liwen Zhang, YingLei Wang, Xiaode Liu, Xinyi Tong, Yuanyuan Ou, Xuhui Huang, and Zhe Ma. Reducing information loss for spiking neural networks. In *ECCV*, 2022. 2, 8

[23] Yufei Guo, Xinyi Tong, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Zhe Ma, and Xuhui Huang. Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In *CVPR*, 2022. 2, 4, 5, 8

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

[25] 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

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

[27] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In *ICML*, 2021. 3

[28] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020. 3

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[30] 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. 6

[31] 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, 8

[32] Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. In *NeurIPS*, 2021. 2, 4, 8

[33] Yuhang Li, Youngeun Kim, Hyoungseob Park, Tamar Geller, and Priyadarshini Panda. Neuromorphic data augmentation for training spiking neural networks. In *ECCV*, 2022. 2

[34] Gehua Ma, Rui Yan, and Huajin Tang. Exploiting noise as a resource for computation and learning in spiking neural networks. *arXiv preprint arXiv:2305.16044*, 2023. 2

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

[36] Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *CVPR*, 2022. 3

[37] Qingyan Meng, Shen Yan, Mingqing Xiao, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training much deeper spiking neural networks with a small number of time-steps. *Neural Networks*, 153:254–268, 2022. 2

[38] 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

[39] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *TNNLS*, 29(7):3227–3235, 2017. 3

[40] 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

[41] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *NeurIPS*, 2016. 3

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 7

[43] 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

[44] Nicolas Perez-Nieves and Dan Goodman. Sparse spiking gradient descent. In *NeurIPS*, 2021. 3

[45] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019. 6

[46] Nitin Rathi and Kaushik Roy. Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *TNNLS*, 2021. 2, 4

[47] 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. 2

[48] 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

[49] 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

[50] Sumit Bam Shrestha and Garrick Orchard. SLAYER: spike layer error reassignment in time. In *NeurIPS*, 2018. 2

[51] Johannes C. Thiele, Olivier Bichler, and Antoine Dupret. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. In *ICLR*, 2020. 3

[52] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 2

[53] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. 3

[54] 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. 3

[55] 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. 3

[56] 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, 4

[57] 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, 4

[58] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. In *NeurIPS*, 2022. 3, 4, 5, 7, 8

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

[60] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Spide: A purely spike-based method for training feedback spiking neural networks. *Neural Networks*, 161:9–24, 2023. 3

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

[62] Qu Yang, Jibin Wu, Malu Zhang, Yansong Chua, Xinchao Wang, and Haizhou Li. Training spiking neural networks with local tandem learning. *arXiv preprint arXiv:2210.04532*, 2022. 3, 7, 8

[63] Bojian Yin, Federico Corradi, and Sander M Bohte. Accurate online training of dynamical spiking neural networks through forward propagation through time. *arXiv preprint arXiv:2112.11231*, 2021. 3, 7

[64] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018. 3

[65] Friedemann Zenke and Emre O Neftci. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE*, 109(5):935–950, 2021. 3, 6

[66] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021. 2

[67] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *AAAI*, 2021. 2, 4, 6, 8

[68] 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. 3