Learnable Surrogate Gradient for Direct Training Spiking Neural Networks

Shuang Lian\textsuperscript{1}, Jiangrong Shen\textsuperscript{1}, Qianhui Liu\textsuperscript{2}, Ziming Wang\textsuperscript{1}, Rui Yan\textsuperscript{3}, Huajin Tang\textsuperscript{1,4}\textsuperscript{*}
\textsuperscript{1}College of Computer Science and Technology, Zhejiang University
\textsuperscript{2}Department of Electrical and Computer Engineering, National University of Singapore
\textsuperscript{3}College of Computer Science and Technology, Zhejiang University of Technology
\textsuperscript{4}Zhejiang Lab
\{slian, jrshen\}@zju.edu.cn, qhliu@nus.edu.sg, zi\_ming\_wang@zju.edu.cn, ryan@zjut.edu.cn, htag@zju.edu.cn

Abstract

Spiking neural networks (SNNs) have increasingly drawn massive research attention due to biological interpretability and efficient computation. Recent achievements are devoted to utilizing the surrogate gradient (SG) method to avoid the dilemma of non-differentiability of spiking activity to directly train SNNs by backpropagation. However, the fixed width of the SG leads to gradient vanishing and mismatch problems, thus limiting the performance of directly trained SNNs. In this work, we propose a novel perspective to unlock the width limitation of SG, called the learnable surrogate gradient (LSG) method. The LSG method modulates the width of SG according to the change of the distribution of the membrane potentials, which is identified to be related to the decay factors based on our theoretical analysis. Then we introduce the trainable decay factors to implement the LSG method, which can optimize the width of SG automatically during training to avoid the gradient vanishing and mismatch problems caused by the limited width of SG. We evaluate the proposed LSG method on both image and neuromorphic datasets. Experimental results show that the LSG method can effectively alleviate the blocking of gradient propagation caused by the limited width of SG when training deep SNNs directly. Meanwhile, the LSG method can help SNNs achieve competitive performance on both latency and accuracy.

1 Introduction

Spiking neural networks (SNNs) are promising for energy-efficient computation under the asynchronous and sparse event-based manner. SNNs process spatio-temporal information with discrete spikes, which is highly compatible with neuromorphic [Davies et al., 2018] and FPGA devices [Ju et al., 2020]. However, due to the non-differentiable spiking activity, it still remains challenges to train high-performance deep SNNs.

There are two main methodologies to address the training problems to avoid the dilemma of non-differentiability. One is the conversion-based method, which converts the pre-trained Convolutional neural networks (CNNs) to SNNs with the same architecture [Hu et al., 2021; Kundu et al., 2021; Wang et al., 2022b], which usually requires considerable timesteps to obtain similar information representation and exists inevitable accuracy loss compared to the original CNNs. Though these methods enable SNNs to achieve comparable performance, the demand for vast timesteps would lead to a large inference latency and high energy consumption problem. Another is the direct training method using the surrogate gradient (SG). This method replaces the all-or-nothing gradients of the spike activity function with different shapes of SG [Wu et al., 2018; Fang et al., 2021a; Deng et al., 2022] to enable gradient backpropagating within a given wider range of membrane potentials. Though this method is promising for training deep SNNs with competitive performance under low latency, it also suffers from the problem of gradient vanishing or explosion, which causes performance degradation and limits to relatively shallow network architectures. Many works [Zheng et al., 2021; Fang et al., 2021a; Feng et al., 2022; Guo et al., 2022b] have been made to solve these problems and prompt the directly trained SNNs to achieve performance improvement.

Nevertheless, due to the lack of comprehensive analysis of the paradigm and difficulty of training SNNs with SG, there still remains improvement in recent works. On the one hand, the limited width of SG causes membrane potentials of numerous of neurons to fall into the saturation area where the approximate derivative is zero or a tiny value, which leads to the gradient vanishing problem. On the other hand, simply setting the width of SG to a large value is inappropriate. In this case, the gradient-available interval will contain values with a large difference, which will cause the gradient mismatch problem and enlarge the approximated errors from the accurate gradients. A proper width of the SG can benefit the direct training of deep SNNs.

To this end, we propose a novel perspective to design SG based on analysing the correlation between the width of SG and the distribution of membrane potential. As the SG is used as a function to determine which membrane potentials have gradients, we first analyze the distribution of the membrane potential in forward propagation. Then we identify that

\textsuperscript{*}Corresponding author
the distributions of membrane potential are related to the decay factors when given the distribution of pre-synaptic input. With the change of decay factors, we can accordingly modulate the width of SG. When we set the decay factors to trainable parameters and optimize them during training, the width of SG can be regarded as learnable equivalently, which is referred as the learnable surrogate gradients (LSG) method in this work. We evaluate the LSG method on both standard image and neuromorphic benchmarks. Experimental results show that the LSG method can alleviate the blocking of gradient propagation resulted by the limited width of SG. Meanwhile, the LSG can help deep SNNs achieve comparable performance on both latency and accuracy.

2 Related Work

2.1 Training Algorithms of SNNs

Conversion. The conversion method is to convert a pretrained CNNs to spiking architecture ([Sengupta et al., 2019; Hu et al., 2021; Han et al., 2020; Yu et al., 2021]) by adjusting parameters of SNN based on the mapping between the activation values of ANN and spike rates of SNNs in a layer-wise manner, which avoids the non-differentiability of SNN training. The major drawback of conversion method is high inference latency and energy consumption. Since then, some techniques have been proposed to reduce the inference latency, i.e., the spike-norm [Sengupta et al., 2019], the channel-wise normalization [Kim et al., 2020], the tandem learning [Wu et al., 2021], the dual-phase error optimization [Wang et al., 2022] and so on. But conversion methods still suffer from the ultra-high latency and could not exploit the temporal dynamics of SNNs sufficiently, thus limiting the flexible application of SNNs.

Direct Training. Direct training methods have been developed rapidly in recent years, and the most popular one is based on the error backpropagation (BP) algorithm with the surrogate gradient. In this way, SNNs are treated as recurrent neural networks and trained with backpropagation through time (BPTT) [Neftci et al., 2019] to propagate gradients through spatial and temporal domains iteratively. Direct training methods are promising for comparable performance under ultra-low latency. In the last few years, this kind of method realizes from spatial error BP [Haeng et al., 2016] to spatial-temporal error BP [Wu et al., 2019; Shrestha and Orchard, 2018; Gu et al., 2019; Zhang and Li, 2020] and has reported high performance on image and neuromorphic datasets. Recent efforts aim to achieve better performance and deeper network structures. [Zheng et al., 2021] proposed the STBP-tDBN method to balance the dynamics of spiking neurons and modified the shortcut connection in standard residual architectures, which enables direct training of very deep SNNs on large-scale datasets. [Li et al., 2021a] quantitatively analyzed the gap between the real gradients and the surrogate gradients during training SNNs and proposed a new kind of spiking neuron model to smooth the gradient estimation by optimizing the shape of the surrogate gradient function with finite difference method adaptively. [Feng et al., 2022] proposed a multi-level firing method based on the STBP method to enable more efficient gradient propagation and the incremental expression ability of the neurons. [Guo et al., 2022b] attempted to rectify the membrane potential distribution and penalizes the undesired shifts during training to reduce the quantization errors.

2.2 Surrogate Gradient Design

The non-differentiability of spiking activity $\frac{\partial o}{\partial u}$ hinders the application of backpropagation for training SNNs directly. Surrogate gradients (SG) are proposed to overcome this challenge, which replaces the derivative of non-differential spike activation and enable gradients to pass within a wider range of membrane potentials. We investigate existing shapes of SG (see Figure 1) as follows: (1) the rectangular [Wu et al., 2018], (2) the exponential [Shrestha and Orchard, 2018], (3) the triangular [Deng et al., 2022], (4) the derivative of a tanh function [Fang et al., 2021] and (5) the derivative of a sigmoid function [Zenke and Vogels, 2021]. Though experiments have shown that the training of SNN is robust to the shape of SG function, a suitable hyperparameter criteria of the SG function, such as the dampening or sharpness of the shapes of SG [Zenke and Vogels, 2021], plays the key role in direct training of deep SNNs [Hajenaars et al., 2021]. Furthermore, optimizing the width (or temperature) of the SG has been proved to be an effective way to improve the learning of deep SNNs [Li et al., 2021a; Leng et al., 2022]. Inspired by previous works, we make efforts to explore how to optimize the parameters of SG during training to achieve superior performance.

2.3 Extended Learnable Parameter in SNNs

To capture the complex dynamics of SNNs during training effectively, many works have taken efforts to bring in additional learnable parameters in SNNs. [Rathi and Roy, 2021] proposed the Diet-SNN to optimize the membrane leak and the firing threshold jointly. [Fang et al., 2021b] presented the parametric Leaky Integrate-and-Fire (PLIF) neuron to set the decay factor to a learnable parameter rather than an empirical hyperparameter. [Wang et al., 2022a] proposed the learnable initial membrane potential mechanism to enable flexible neuronal mechanisms across layers. Nevertheless, previous works mainly focus on the learnable neuronal dynamics, and
to our best knowledge, there is no systematic research on how to set the shape of SG learnable, which is quite crucial for training SNNs directly. To this end, we propose an effective way to design the learnable SG to achieve superior performance for direct training SNNs.

3 Preliminaries

3.1 Spiking Neuron Model

We adopt the iterative leaky integrate-and-fire (LIF) model [Wu et al., 2018] as the basic computational unit in SNN. The state updating equations are as follows

\[
I_{i}^{n+1,t+1} = \sum_{j=1}^{L(n)} w_{ij}^{n} o_{j}^{n,t+1}
\]

(1)

\[
u_{i}^{n+1,t+1} = \beta \nu_{i}^{n+1,t} (1 - o_{i}^{n+1,t}) + I_{i}^{n+1,t+1}
\]

(2)

\[o_{i}^{n+1,t+1} = \Theta(u_{i}^{n+1,t+1} - v_{th})
\]

(3)

where the subscripts \(n, t, i\) indicate that the state of the \(i\)-th neuron in the \(n\)-th layer at the \(t\)-th time point. \(L(n)\) denotes the number of neurons in \(n\)-th layer. \(I\) is the pre-synaptic inputs, \(u\) means the membrane potential, \(\beta\) is the decay factor. \(w_{ij}\) is the synaptic weight from the \(j\)-th neuron in pre-layer \((n)\) to the \(i\)-th neuron in the post-layer \((n+1)\). \(\Theta(\cdot)\) is the spike function, which satisfies \(\Theta(x) = 0\) when \(x < 0\), otherwise \(\Theta(x) = 1\). When the membrane potential \(u\) reaches the threshold \(v_{th}\), the neuron will fire a spike and \(u\) is reset to 0 for simplicity.

3.2 Surrogate Gradients Learning in SNNs

The most popular shape of SG is the rectangular function [Wu et al., 2019; Zheng et al., 2021; Deng et al., 2022], defined by

\[
\frac{\partial o_{i}^{n,t}}{\partial u_{i}^{n,t}} \approx h(u_{i}^{n,t}) = \frac{1}{\alpha} \operatorname{sign}(|u_{i}^{n,t} - v_{th}| - \frac{\alpha}{2})
\]

(4)

where \(\alpha\) is a hyperparameter to determine the width of \(h(\cdot)\) and is fixed in all previous works. The gradient-available interval is \([v_{th} - \frac{\alpha}{2}, v_{th} + \frac{\alpha}{2}]\). In this case, if the width \(\alpha\) is set too large, the gradient-available interval will contain values with a large difference, which will cause the gradient mismatch problem and enlarge the approximated errors from the accurate gradients. On the other hand, if we select a small width, numerous spiking neurons will fall into the saturation area outside the rectangular area, and the corresponding \(\frac{\partial o_{i}^{n,t}}{\partial u_{i}^{n,t}}\) will be zero due to the limited width of SG as shown in Figure 2. Hence, the gradients of these neurons will be lost, which leads to the gradient vanishing and hinders the training of deep SNNs. So how to choose an appropriate width of \(h(\cdot)\) is essential for training deep SNNs directly [Li et al., 2021b], which is the main motivation of our work.

3.3 Threshold-Dependent Batch Normalization

The batch normalization [Ioffe and Szegedy, 2015] (BN) technique can accelerate training and reduce internal covariate shift during the optimization of ANNs. But the BN layers are not designed for normalizing spatial-temporal data and directly transplanting the BN technique in SNN training may lead to undesired results. To this end, [Zheng et al., 2021] modified the feedforward form of the temporal domain and proposed the threshold-dependent batch normalization (tdBN) method to normalize the pre-synaptic inputs \(I\) in both spatial and temporal domains to make the BN technique support spatial-temporal information processing. Let \(I_{k}^{t}\) represents the \(k\)-th channel feature maps of \(I^{t}\), and \(I_{k} = (I_{k}^{1}, I_{k}^{2}, \ldots, I_{k}^{T})\) will be normalized as

\[
\hat{I}_{k} = \frac{E[I_{k} - E[I_{k}]]}{\sqrt{\text{var}[I_{k}]} + \epsilon}
\]

(5)

\[
T_{k} = \gamma \hat{I}_{k} + \mu
\]

(6)

where \(E\) and \(\text{var}\) compute the mean and variance in channel dimension, respectively. \(\gamma\) and \(\mu\) are learnable parameters. \(\eta\) is a wisely chosen hyperparameter to prevent over-fire and under-fire. The normalized input after tdBN \(T_{k}\) will be fed into Eq.2. We adopt the tdBN method to normalize the pre-synaptic inputs to a normal distribution with a mean of 0.

4 Method

In this section, we will introduce the design details of the learnable surrogate gradients (LSG) learning and the whole training process.

4.1 Analysis of Membrane Potential’s Dynamics

Since the SG method can be seen as an approximate function for the membrane potential \(u\), we start with the analysis of dynamics about \(u\).

In the forward propagation, the pre-synaptic input \(I\) is normalized by the threshold-dependent batch normalization (tdBN) [Zheng et al., 2021] method and satisfies \(I \sim N(0, \sigma_{I}^{2})\). Based on that, we propose Theorem 1 to explain the detailed dynamics of membrane potential.

**Theorem 1.** With the iterative LIF model and the tdBN method, assuming the pre-synaptic input \(I \sim N(0, \sigma_{I}^{2})\), we have the membrane potential \(u \sim N(0, \sigma_{mem}^{2})\) and \(\sigma_{mem}^{2} = g(\beta) * (v_{th})^{2}\), wherein the \(g(\beta)\) means a directly proportional function of \(\beta\) and \(g(\beta)\) can be approximated as \((1 + \beta^{2})\).

**Proof.** The proof of Theorem 1 is presented in Supplementary Material A.
We verify Theorem 1 by visualization analysis. In the experiment, we set the pre-synaptic input \( I \sim N(0, (0.5)^2) \) and display the distribution of membrane potential of LIF neurons with different decay factors. As shown in Figure 3, when given fixed pre-synaptic inputs, neurons with different decay factors have different variances, which supports the proposition.

Theorem 1 explains the relation between the decay factors and the distributions of membrane potential. Given the normalized pre-synaptic input \( I \sim N(0, v_{th}^2) \), the distributions of membrane potential are only determined by the decay factor. Since the width of SG \( \alpha \) in Eq.(4) determines which values of membrane potential have gradients (see Figure 3), once the decay factor \( \beta \) as well as the distributions of membrane potential changed, we can accordingly modulate the width of SG to avoid the problem raised by the limited width \( \alpha \). It is obvious that there exists a correlation between \( \alpha \) and \( \beta \). We can formulate a function \( f(\cdot) \) to describe this correlation precisely, namely \( \alpha = f(\beta) \). As a result, if we consider \( \beta \) as a trainable parameter [Fang et al., 2021b], the width of SG can be also learned during training, which is referred as the learnable surrogate gradients (LSG) method.

4.2 Design of Learnable Surrogate Gradients

Based on our analysis, we can identify the basic rules on the design of LSG learning, which are listed as follows:

1. The decay factor \( \beta \) of LIF neuron is set as a trainable parameter and can be optimized automatically during training, rather than a fixed constant.
2. The function \( f(\cdot) \) needs to be set as a directly proportional function of the decay factor \( \beta \) manually before training.
3. Neurons in the same layer in SNNs share one decay factor, and neurons in different layers have different decay factors. That means the values of \( f(\beta) \) would be distinct in different layers during training, which reshapes the layer-wise width of SG in the training process.

With the three rules, we design the LSG in two steps. First, we follow previous works [Fang et al., 2021b] and adopt a trainable parameter \( b \) to formulate the decay factor \( \beta \) instead of the direct optimizing, which is described as

\[
\beta_n = k(b_n) = \frac{1}{1 + e^{-b_n}} \tag{7}
\]

where \( k(\cdot) \) is the clamp function to ensure \( \beta \in (0, 1) \) and \( n \) indicates the \( n \)-th layer. \( \beta_n \) is initialized to 0.2 for all layers. So, the decay factor \( \beta \) can be optimized automatically during training in a layer-wise manner.

Second, we set the \( \alpha = f(\beta_n) = 2 \cdot v_{th} \sqrt{1 + \beta_n^2} \), so Eq.(4) can be rewritten as

\[
h(u_i^{n,t}) = \frac{1}{2 \cdot v_{th} \sqrt{1 + \beta_n^2}} \text{sign}(u_i^{n,t} - v_{th} < v_{th} \sqrt{1 + \beta_n^2}) \tag{8}
\]

which means that the gradient-available interval is \( [v_{th} - v_{th} \sqrt{1 + \beta_n^2}, v_{th} + v_{th} \sqrt{1 + \beta_n^2}] \) and can be adjusted with the optimization of \( \beta \) during training. We formalize this improvement in Theorem 2.

Theorem 2. With the empirical experimental setting \( \alpha = 1 \) and \( v_{th} = 0.5 \), assume the possibilities of a neuron leading to gradient vanishing with and without LSG learning are \( P^* \) and \( P \) respectively, then we have \( P^* < P \).

Proof. The proof of Theorem 2 is presented in Supplementary.

In this way, the LSG learning can effectively optimize the width of the SG during training, so as to further avoid the problems caused by the limited width of SG.

In conclusion, based on our analysis, we confirm the direct proportionality relation between the decay factor and the distribution of membrane potential when given the pre-synaptic, and we can modulate the width of SG according to the distribution of membrane potential. Thus, we can consider the width of SG as a function of the decay factor, and when we set the decay factor to a trainable parameter, the width of SG is treated as learnable during training.

4.3 Overall Training Process

In this section, we present the overall training process for deep SNNs with the LSG learning and STBP algorithm [Wu et al., 2019] in the output layer, instead of firing them across time, we choose to integrate the output as did in recent works [Zheng et al., 2021; Li et al., 2021a]. The accumulated membrane is described as follows

\[
u_i = \frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{L(N-1)} w_{ij} o_{j,t}^{n,t}, i \in \{1, 2, \ldots, c\} \tag{9}\]

where \( T \) is the length of timestep and \( c \) is the number of neurons in the output layer, which is equal to the number of sample classes. Then, we can compute the cross-entropy loss based on the true label and the output accumulated membrane of SNN.

With the accumulated membrane of the output layer \( u = (u_1, u_2, \ldots, u_c) \) and label vector \( Y = (y_1, y_2, \ldots, y_c) \), the loss function is determined by the cross-entropy function, which is described as

\[
p_i = \frac{e^{u_i}}{\sum_{j=1}^{c} e^{u_j}} \tag{10}\]

\[
L = - \sum_{i=1}^{c} y_i \log(p_i) \tag{11}\]
Algorithm 1 Overall training process of the SNN with LSG method in one iteration

**Input:** Timestep: $T$; Threshold: $v_{th}$; Initial layer-wise decay: $\beta_n$; input $o = o^1, o^2, ..., o^T$; label $Y$.

**Output:** updated $w_{ij}^n$ and $b_n$ of the SNN.

**Forward:**

1. for $n = 1$ to $N$ do
2. for $t = 1$ to $T$ do
3. if $n < N$ then
4. $I_{n+1,t} = w_{ij}^n \circ o^{n,t}$, // Eq.(1)
5. else
6. $u^N = \text{Accumulate}(o^{N-1,t})$ // Eq.(9)
7. end if
8. end for
9. $T' \leftarrow \text{tdBN}(I^n)$ // Eq.(5) and Eq.(6)
10. Calculate the spike output $o^{t+1}$ // Eq.(2) and Eq.(3)
11. Calculate the width of SG $f(\beta_n)$
12. end for
13. $L \leftarrow \text{CrossEntropy}(u^N, Y)$ // Eq.(10) and Eq.(11)

**Backward:**

14. for $n = N$ to 1 do
15. for $t = T$ to 1 do
16. $\frac{\partial L}{\partial o_i^{n,t}} \leftarrow \text{GradBackward}(\frac{\partial L}{\partial o_j^{n+1,t}}, \frac{\partial L}{\partial o_i^{n,t+1}})$ // Eq.(12)
17. $\frac{\partial L}{\partial u_i^{n,t}} \leftarrow \text{GradBackward}(\frac{\partial L}{\partial o_i^{n,t}}, \frac{\partial L}{\partial u_i^{n,t+1}}, f(\beta_n))$ // Eq.(8) and Eq.(13)
18. end for
19. end for
20. Update parameters $w_{ij}^n$ and $b_n$. // Eq.(14), Eq.(15)

end for

where $c$ means the number of classes.

With the STBP algorithm, the gradients can be computed by

$$\frac{\partial L}{\partial o_i^{n,t}} = \sum_{j=1}^{L(n+1)} \frac{\partial L}{\partial o_j^{n+1,t}} \frac{\partial o_j^{n+1,t}}{\partial d_j^{n+1,t}} + \frac{\partial L}{\partial u_i^{n,t+1}} \frac{\partial u_i^{n,t+1}}{\partial o_i^{n,t}} \tag{12}$$

$$\frac{\partial L}{\partial u_i^{n,t}} = \frac{\partial L}{\partial o_i^{n,t}} \frac{\partial o_i^{n,t}}{\partial u_i^{n,t}} + \frac{\partial L}{\partial u_i^{n,t+1}} \frac{\partial u_i^{n,t+1}}{\partial u_i^{n,t}} \tag{13}$$

where $o^{n,t}$ and $u^{n,t}$ represent the spike and membrane potential of the neuron in $n$-th layer at $t$-th time point. Finally, we can obtain the gradients of weights $w_{ij}^n$ and the trainable parameter $b_n$ as

$$\frac{\partial L}{\partial w_{ij}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_{ij}^n} \frac{\partial u_{ij}^n}{\partial I_{ij}^n} \tag{14}$$

$$\frac{\partial L}{\partial b_n} = \sum_{t=1}^{T} \frac{\partial L}{\partial u_i^{n,t}} \frac{\partial u_i^{n,t}}{\partial b_n} \tag{15}$$

With Eq.(10) - Eq.(15), the gradients can be backpropagated along both spatial and temporal domains. Details of the training algorithm with the LSG method are shown in Algorithm 1.

---

**Table 1:** Ablation Study for LSG learning on different datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>None</td>
<td>92.68%</td>
</tr>
<tr>
<td></td>
<td>w/ trainable decay</td>
<td>93.16%</td>
</tr>
<tr>
<td></td>
<td>w/ LSG</td>
<td>94.41%</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>None</td>
<td>73.87%</td>
</tr>
<tr>
<td></td>
<td>w/ trainable decay</td>
<td>74.12%</td>
</tr>
<tr>
<td></td>
<td>w/ LSG</td>
<td>76.22%</td>
</tr>
<tr>
<td>CIFAR-DVS</td>
<td>None</td>
<td>75.40%</td>
</tr>
<tr>
<td></td>
<td>w/ trainable decay</td>
<td>77.50%</td>
</tr>
<tr>
<td></td>
<td>w/ LSG</td>
<td>77.50%</td>
</tr>
</tbody>
</table>

**Table 2:** Comparison results with different width of SG on CIFAR-10 dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG $(\alpha = 0.5)$</td>
<td>92.12%</td>
</tr>
<tr>
<td>SG $(\alpha = 1.0)$</td>
<td>92.68%</td>
</tr>
<tr>
<td>SG $(\alpha = 2.5)$</td>
<td>93.16%</td>
</tr>
<tr>
<td>SG $(\alpha = 5.0)$</td>
<td>93.71%</td>
</tr>
<tr>
<td>SG $(\alpha = 10.0)$</td>
<td>94.41%</td>
</tr>
</tbody>
</table>

**Table 5:**

5. **Experiments**

We evaluate our work on both image datasets (CIFAR-10/100) and the neuromorphic dataset, CIFAR-DVS [Li et al., 2017]. We first conduct a series of ablation experiments to verify the effectiveness of the proposed LSG method. Then we explore how the LSG method alleviates the blocking of gradient propagation during training. We finally compare our LSG method with previous methods to illustrate the superiority of our work. Details of the hyperparameter settings and the network structures are introduced in Supplementary.

5.1 **Ablation Study**

We conduct a set of ablation experiments to verify the effectiveness of the proposed LSG learning on CIFAR-10/100 using ResNet-19 [Zheng et al., 2021] with $T=2$ and CIFAR-DVS using VGGSNN [Deng et al., 2022] with $T=10$ as backbones.

Detailed results of different SNNs are illustrated in Table 1. On the CIFAR-10 dataset, SNN with the LSG achieves 94.89% accuracy, surpassing the vanilla one and SNN with the trainable decay by 1.73% and 1.25% respectively. On the CIFAR-100 dataset, SNN with the LSG achieves 76.22% accuracy, which is significantly better than SNNs trained without any method and with the trainable decay. The proposed LSG learning method also demonstrates its superiority on the CIFAR-DVS dataset. The performance improvements brought by the LSG method are significant. We can conclude the effectiveness of the LSG method for training deep SNNs. Besides, We record the testing accuracy and loss of different SNNs on CIFAR-10/100 dataset during training. As illustrated in Figure 4, the LSG learning method can not only help SNN achieve better results on both datasets, but also accelerate the convergence speed of network training.
Figure 4: Recordings of testing accuracy (top row) and loss (bottom row) when training on CIFAR-10/100 datasets with 2 timesteps.

Figure 5: The proportion of spiking neurons falling into the gradient-available interval in each convolutional layer.

We also compare the LSG method with SG method having different widths on CIFAR-10 with ResNet-19 under $T=2$. As shown in Table 2, we can see that the width of SG greatly affects the performance of SNNs, and it is catastrophic damage for SG when the width $\alpha$ is selected inappropriately ($\alpha \geq 5$), which is consistent with our analysis. In contrast, the LSG method helps SNN achieve the best accuracy, which verifies that the LSG can effectively optimize the width of SG for better performance.

5.2 LSG for Gradient Vanishing

In this part, we conduct a series of experiments to demonstrate that the LSG method can alleviate the blocking of gradient propagation resulted by the limited width of SG. We apply the ResNet-19 on CIFAR-10 with 2 timesteps.

We first visualize the proportion of spiking neurons of different SNNs falling into the gradient-available interval in each layer in Figure 5. We can see that there is a large proportion of spiking neurons falling out of the gradient-available interval when SNN is trained without any method, especially as the layer deepens, which will lead to gradient vanishing. And the trainable decay method can not alleviate this situation. For example, in 11th and 16th layers, SNN trained without any method has a higher proportion than the SNN trained with the trainable decay method. In contrast, while trained with the LSG method, SNN can jump out of the dilemma and ensures a certain proportion of spiking neurons falling into the gradient-available interval to prevent gradient vanishing, which shows the potential to train deep SNNs.

To further explore how the LSG method helps SNN jump out of the dilemma, we record the change curves of $f(\beta_n)$ during training. We select the 2nd, 6th, 10th, and 16th layers, for their significantly high proportion of spiking neurons falling into the gradient-available interval. As shown in Figure 6, $f(\beta_n)$ of different layers have different curves and finally converge to different values. It is worth noting that the $f(\beta_n)$ tends to increase as the layer goes deeper, which does not occur in SNN trained with trainable decay as shown in Figure 7. That indicates the LSG method helps SNN to maintain enough spiking neurons having gradients in the deep-layer and alleviate the gradient vanishing problem.

5.3 Comparisons with Other Methods

In this section, we compare our experimental results with previous works on image datasets and neuromorphic dataset. Results are illustrated in Table 3. Details of data preprocessing are introduced in Supplementary. All the experimental results are averaged over 5 runs.

For CIFAR-10, based on the ResNet-19, our SNN trained with LSG method achieves 95.52% accuracy with only 6 timesteps and surpasses all the other compared meth-
ods. We can notice that under ultra-low latency ($T=2$), the LSG method could obtain comparable performance (reaching 94.41%) and outperforms the previous best accuracy by a margin of 0.77%.

For CIFAR-100, SNN trained with LSG method based on ResNet-19 achieves the best accuracy of 77.13% with only 6 time steps, which outperforms other recent compared methods with the same network structure by a margin of 2.41%, 1.03% and 0.72% respectively. It is worth noting that under only 2 timesteps, our method still achieves a better result (reaching 76.32%) than most of the compared methods, while compared with other methods, we achieve significantly better result. Further, with the TET loss and the augmentation technique proposed in [Deng et al., 2022], the accuracy rises to 83.70%, which outperforms existing methods by a large margin.

### 6 Conclusion

In this work, we propose the learnable surrogate gradients (LSG) method to unlock the width limitation of SG in direct training SNNs. We first identify the correlation between the distributions of the membrane potentials and the decay factors when given the pre-synaptic inputs based on our theoretical analysis. Thus, we can use this correlation to modulate the width of SG when the decay factors as well as the distributions of the membrane potentials change. When the decay factors are set to trainable parameters, the width of SG can be treated as learnable, which is referred as the learnable surrogate gradients (LSG) method. The LSG method can automatically optimize the width of SG during training and avoid the gradient vanishing and mismatch problems caused by the limited width of SG. Experimental results and analysis show that the LSG method can effectively alleviate the blocking of gradient propagation resulted by the limited width of SG when training deep SNNs directly, and helps SNNs achieve competitive performance on both latency and accuracy.

### Table 3: Comparison results with existing works on different datasets.* denotes using the TET loss and data augmentation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Architecture</th>
<th>Timestep</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>ResNet-19 [Zheng et al., 2021]</td>
<td>ResNet-19</td>
<td>6</td>
<td>93.16%</td>
</tr>
<tr>
<td></td>
<td>Conversion [Hu et al., 2021]</td>
<td>ResNet-44</td>
<td>350</td>
<td>92.37%</td>
</tr>
<tr>
<td></td>
<td>Dspike [Li et al., 2021a]</td>
<td>ResNet-18</td>
<td>6</td>
<td>94.25%</td>
</tr>
<tr>
<td></td>
<td>7-layer CNN</td>
<td>8</td>
<td>93.50%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Deng et al., 2022]</td>
<td>ResNet-19</td>
<td>6</td>
<td>94.50%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Feng et al., 2022]</td>
<td>ResNet-19</td>
<td>4</td>
<td>94.25%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Guo et al., 2022b]</td>
<td>ResNet-19</td>
<td>2</td>
<td>93.64%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Duan et al., 2022]</td>
<td>ResNet-19</td>
<td>6</td>
<td>94.71%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Guo et al., 2022a]</td>
<td>ResNet-19</td>
<td>6</td>
<td>95.49%</td>
</tr>
<tr>
<td>LSG</td>
<td>ResNet-19</td>
<td>6</td>
<td>95.52 ± 0.05%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ResNet-19</td>
<td>4</td>
<td>95.17 ± 0.05%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ResNet-19</td>
<td>2</td>
<td>94.41 ± 0.08%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Architecture</th>
<th>Timestep</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-100</td>
<td>ResNet-19 [Deng et al., 2022]</td>
<td>ResNet-19</td>
<td>6</td>
<td>77.90 ± 0.15%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Deng et al., 2022]</td>
<td>ResNet-19</td>
<td>4</td>
<td>76.85 ± 0.10%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Deng et al., 2022]</td>
<td>ResNet-19</td>
<td>2</td>
<td>76.32 ± 0.12%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>Architecture</th>
<th>Timestep</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7-layer CNN</td>
<td>20</td>
<td>74.80%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ResNet-18 [Deng et al., 2022]</td>
<td>ResNet-19</td>
<td>6</td>
<td>75.45%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Feng et al., 2022]</td>
<td>ResNet-19</td>
<td>10</td>
<td>77.40%</td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Guo et al., 2022b]</td>
<td>ResNet-19</td>
<td>10</td>
<td>72.42%</td>
</tr>
<tr>
<td></td>
<td>7-layer CNN</td>
<td>10</td>
<td>75.10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ResNet-19 [Duan et al., 2022]</td>
<td>ResNet-19</td>
<td>10</td>
<td>72.60%</td>
</tr>
<tr>
<td>LSG</td>
<td>ResNet-19</td>
<td>10</td>
<td>83.70* ± 0.15%</td>
<td></td>
</tr>
</tbody>
</table>

[3008]
Acknowledgments

This work was supported by National Key Research and Development Program of China under Grant No. 2020AAA0105900, National Natural Science Foundation of China under Grant No. 62236007, Key Research and Development Program of Zhejiang under Grant No. 2023C03001 and A*STAR under its RIE2020 IAF-ICP grant I2001E0053.

References


