Surrogate Module Learning: Reduce the Gradient Error Accumulation in Training Spiking Neural Networks

Shikuang Deng [1 2] Hao Lin [1] Yuhang Li [3] Shi Gu [1 2]

Abstract

Spiking neural networks (SNNs) provide an alternative solution to conventional artificial neural networks with energy-saving and high-efficiency characteristics after hardware implantation. However, due to its non-differentiable activation function and the temporally delayed accumulation in outputs, the direct training of SNNs is extraordinarily tough even adopting a surrogate gradient to mimic the backpropagation. For SNN training, this non-differentiability causes the intrinsic gradient error that would be magnified through layerwise backpropagation, especially through multiple layers. In this paper, we propose a novel approach to reducing gradient error from a new perspective called surrogate module learning (SML). Surrogate module learning tries to construct a shortcut path to back-propagate a more accurate gradient to a certain SNN part utilizing the surrogate modules. Then, we develop a new loss function for concurrently training the network and enhancing the surrogate modules' surrogate capacity. We demonstrate that when the outputs of surrogate modules are close to the SNN output, the fraction of the gradient error drops significantly. Our method consistently and significantly enhances the performance of SNNs on all experiment datasets, including CIFAR-10/100, ImageNet, and ES-ImageNet. For example, for spiking ResNet-34 architecture on ImageNet, we increased the SNN accuracy by 3.46%. Codes are available at https://github.com/Gus-Lab/temporal_efficient_training.

[*]Equal contribution [1]University of Electronic Science and Technology of China [2]Shenzhen Institute for Advanced Study, UESTC [3]Yale University. Correspondence to: Shi Gu <gus@uestc.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Inspired by biological networks, the spike neural network (SNN) selects the binary spike signal as the information carrier, giving it the potential for high speed and low energy consumption (Maass, 1997; Roy et al., 2019). The neuron in SNN activates and fires, which is emitted only when the membrane potential exceeds the threshold. Spikes can be expressed numerically as 0 and 1, where 0 indicates the neuron is at rest and 1 indicates the neuron is active. The input of the network between each layer is only 0 or 1, so the high-precision multiplication operation of the matrix can degenerate into a high-precision addition operation. When embedded in the neuromorphic hardware (DeBole et al., 2019; Davies et al., 2018), this characteristic significantly increases the computing speed of the SNN and reduces the network's energy consumption (Shrestha & Orchard, 2018; Kim et al., 2019).

Although the inclusion of binary spiking signals gives SNNs the potential for high speed and low energy consumption, the discrete nature of the activation function makes training SNNs extremely challenging. In a traditional artificial neural network (ANN), the update gradient of the network weights is calculated by a back-propagation algorithm, which is feasible for continuous numerical signals. Nevertheless, due to the discrete nature of the spike signal in SNNs, the back-propagation technique cannot be directly used to compute the gradient. The surrogate gradient (SG) technique is a common solution to this problem (Lee et al., 2016; Wu et al., 2018; Zheng et al., 2021). The surrogate gradient method makes gradient computation feasible by replacing the non-derivable step function (spike activate function) with a smooth curve. Although the surrogate gradient algorithm allows us to flexibly design the network structure and requires few simulation time steps (Deng et al., 2021), on deep architectures, the performance of SNNs is still lower than that of ANNs due to the surrogate gradient error accumulation. Some recent works have emerged to design or search for a better surrogate gradient shape to reduce the gradient error (Li et al., 2021; Herranz-Celotti & Rouat, 2022; Suetake et al., 2022). These researches have enhanced the SNN performance, but they still have a gap between ANN. And under the same network architecture, the performance gap between ANN and SNN increases with network depth (Hu et al., 2021).

While onsite backpropagation is not well supported by the brain anatomy, the feedback connections from high-level brain modules, e.g. the frontoparietal network to the sensory and executive modules do exist (Friston, 2003) and may deliver error information to the pointed modules(Lillicrap et al., 2020). Although the exact role of these feedback connections remains unclear, we hypothesize that it may enable long-range error propagation for effective multi-layer and multi-time-point learning. In this study, we provide a surrogate module learning approach that provides a new gradient backward path, so mitigating the gradient error accumulation problem caused by the surrogate gradient. When a layer inserts a surrogate module, the gradient it obtains has two parts: one back-propagate from the subsequent layers of the SNN and one back-propagate from the surrogate module. We prove that introducing a surrogate module will reduce the gradient error caused by the surrogate gradient. It is worth noting that surrogate modules only work during the training phase and can be removed right away when training is completed, i.e., we do not alter the SNN forward-propagation during inference. SNN performance has improved significantly in almost all datasets with surrogate module learning. Fig. 1 depicts the concept of the surrogate module method.

The following summarizes our main contributions:

- We devised a surrogate module learning method to mitigate the effect of surrogate gradient estimation error accumulation.
- We theoretically analyze the feasibility of the surrogate module learning in SNN and provide sufficient experiments to verify the effectiveness of our method. For instance, our method improves the performance of spiking ResNet-34 on ImageNet by 3.46%.
- We demonstrate the possibility of SNN training without layerwise backpropagation and suggests a potential role for backward connections to enable long-range error propagation in deep spiking neural networks.

2. Related Work

SNN Direcct Training. The direct training approach employs a backpropagation-based algorithm like STBP to train the SNN. In this technique, the non-differentiable spike activation function is substituted by a smooth and differentiable function, known as the surrogate gradient. Numerous surrogate gradients, such as exponential (Shrestha & Orchard, 2018), Actan (Fang et al., 2021), S2NN (Suetake et al., 2022), Dspike (Li et al., 2021), etc., have been proposed recently. The selection of a surrogate gradient will influence the training result, and different surrogate gradients may be suitable for different network structures and datasets (Li et al., 2021). Then, with the introduction of the TDBN (Zheng et al., 2021) algorithm, the performance of direct training has been significantly enhanced, and it is now capable of producing superior results even on the ImageNet dataset. Another advantage of direct training is that it is suitable for neuromorphic datasets. Due to the high noise and temporal domain characteristics of neuromorphic datasets, the results of traditional ANN are worse than direct training SNN (Deng et al., 2020). With the introduction of specially designed methods like TET (Deng et al., 2021), TA-SNN (Yao et al., 2021) and TCJA-SNN (Zhu et al., 2022), the performance of SNN on neuromorphic datasets has been further improved. However, the direct training results of SNN on static datasets can still not be compared to those of ANN due to the limitation of the information expression ability of spikes and the effect of gradient error from surrogate gradients.

Auxiliary Module. In the middle of 2010s, the auxiliary module is used to solve the training problem that network is too deep (Szegedy et al., 2015). The auxiliary module exists in the training phase and is deleted when training is finished. Recently, the auxiliary module has been heavily used in local learning (Belilovsky et al., 2020; Duan & Principe, 2021; Belilovsky et al., 2019). Local learning divides the network into many local blocks and stops the gradient backpropagation between them. Instead, each local block will be connected to an auxiliary module for gradient backpropagation and training. A network search study has demonstrated that well-designed local block divisions and auxiliary module structures can make local learning better than end-to-end learning (Pyeon et al., 2020). Meanwhile, a recent work (BYOT) designs different exits that can be seen as auxiliary modules for ANN to balance the inference speed and the network accuracy (Zhang et al., 2019). It designs a self-distillation method to significantly improve the early exits' accuracy and finally enhance the overall network accuracy. The previous works demonstrate that the auxiliary module has the potential to help network training improve performance. There are also some related works in SNN that use auxiliary paths to assist training, for example: DECOLLE (Kaiser et al., 2020) designs a layer-wise local learning for SNN with a readout layer; some work to construct tandem learning, which uses an ANN that shares parameters with SNN to backpropagate gradients (Wu et al., 2021a; Xiao et al., 2021; Meng et al., 2022); or direct learning from a pre-trained ANN layer-by-layer (Yang et al., 2021).

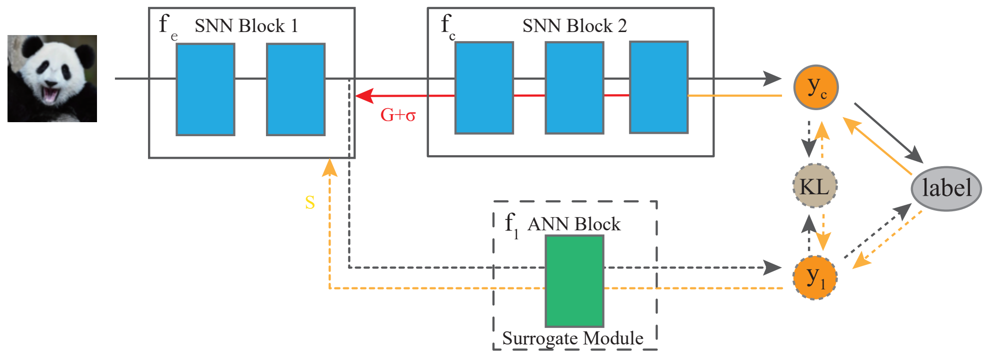

Figure 1. The concept of surrogate module learning. The figure illustrates the situation of using one surrogate module to help SNN training. The SNN is divided into two parts: f_e and f_c. The gradient $(G + \sigma)$ of f_e back propagate from f_c contains gradient error because it has passed through the surrogate gradient. G denotes the correct enough gradient that can be obtained by FDG (Li et al., 2021) or NA (Yang et al., 2021). Surrogate module training seeks to provide an additional path f_1 known as the surrogate module of f_c in order to return a more precise gradient (S) to f_e. It realizes this goal by employing ANN as the surrogate module and approximating SNN f_c's output by self-distillation. After training is complete, the surrogate module-related portion, shown with a dash in the figure, will be eliminated without affecting the SNN inference.

3. Preliminaries

3.1. SNN Neural Model

We use the Leaky Integrate-and-Fire (LIF) model and convert it using the Euler technique into an iterative expression (Wu et al., 2019). Mathematically, the membrane potential is updated as

$$\mathbf{u}(t+1) = \tau\mathbf{u}(t) + \mathbf{I}(t), \tag{1}$$

where τ is the constant leaky factor, $\mathbf{u}(t)$ is the membrane potential at time t, and $\mathbf{I}(t)$ denotes the pre-synaptic inputs, which is the output (pre-activation value) of the convolutional, pooling, or fully connected layer of SNN. When the membrane potential exceed a specific threshold, V_{th}, the neuron fires a spike $\mathbf{a}(t+1)$, which equals to 1. The membrane potential will then follow the hard reset mechanism and return to an initial potential, which is typically 0. So the firing function and hard reset mechanism can be described as

$$\mathbf{a}(t+1) = \mathbf{\Theta}(\mathbf{u}(t+1) - V_{th}) \tag{2}$$

$$\mathbf{u}(t+1) = \mathbf{u}(t+1)\cdot(1-\mathbf{a}(t+1)), \tag{3}$$

where $\mathbf{\Theta}$ denotes the Heaviside step function. We do not search for the optimal neuron model parameters in this paper, but rather use the most common option, which is to set the initial potential $\mathbf{u}(0)$to 0, the threshold V_{th} to 1, and the leaky factor τ to 0.5.

3.2. Surrogate Gradient

Following the concept of direct training, we regard the SNN as an RNN and calculate the gradients through spatial-temporal backpropagation (STBP) (Wu et al., 2018):

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_t \frac{\partial L}{\partial \mathbf{a}(t)} \frac{\partial \mathbf{a}(t)}{\partial \mathbf{u}(t)} \frac{\partial \mathbf{u}(t)}{\partial \mathbf{I}(t)} \frac{\partial \mathbf{I}(t)}{\partial \mathbf{W}}, \tag{4}$$

where the term $\frac{\partial \mathbf{a}(t)}{\partial \mathbf{u}(t)}$ is the gradient of the non-differentiability step function involving the derivative of Dirac δ function that is typically replaced by surrogate gradients with a derivable curve. In this paper, we employ the Dspike (Li et al., 2021) function as the surrogate gradient function, which consists of a tanh function that controls the proximity of the smooth curve to the Delta-function through a temperature coefficient b. Mathematically, Dspike's forward function can be described as:

$$\text{Dspike}(x,b) = \frac{\tanh(b\cdot x)}{2\tanh(r\cdot b)} + \frac{1}{2}, \text{if } -r \le x \le r. \tag{5}$$

where $r = 1$ denotes the sample region.

4. Methodology

4.1. Problem of Surrogate Gradient

The gradient error lies in the difference between the smooth curve of the surrogate gradient and the Dirac δ function. On the one hand, if the surrogate gradient is too close, the Dirac δ function may cause the gradient to vanish and thus fail training. On the other hand, if the surrogate gradient is not close to the Dirac δ function, the gradient error may be very large. Even though there are several studies devoted to optimizing the shape of the surrogate gradient (Li et al., 2021; Herranz-Celotti & Rouat, 2022; Suetake et al., 2022), they still suffer from gradient errors that cannot be completely eliminated. Meanwhile, the gradient error of the surrogate gradient grows as the number of layers and SNN simulation time steps increase, making network convergence harder. Therefore, directly training a deeper SNN with the surrogate gradient usually results in poor performance.

Here, we examine the performance variation of directly trained ANNs and SNNs ($T = 4$) at different depths on CIFAR-10 using a toy model. The toy model begins with three stem layers, for example 3C64s1-3C64s2-3C128s2, where 3C64s1 means using a convolutional layer with a kernel size of 3, output channels 64, and stride 1. Then we connect L 3C128s1 convolutional layers, and finally use an adaptive average pooling layer with the kernel size 2 and a classifier. As shown in Fig. 2, the accuracy peak of ANN occurs at $L = 13$, whereas the accuracy peak of vanilla SNN appears at $L = 5$. And as the depth exceeds 13, the accuracy of vanilla SNN decreases dramatically as the depth continues to increase, While ANN does not demonstrate a substantial accuracy decline. This demonstrates that the gradient error of the surrogate gradient will progressively accumulate with the network depth, and therefore we cannot train a deep SNN just using the surrogate gradient alone.

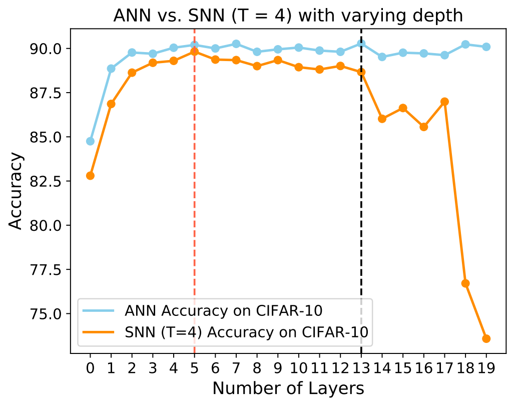

Figure 2. Comparison of the accuracy of ANN and SNN with different depth.

4.2. Surrogate Module Learning

Auxiliary Network. Assuming that the final training goal is to minimize a loss function, $\mathcal{L}$, and the network can be divided into two parts, the first part can be viewed as a feature encoder, f_e, with a trainable parameter, θ_e, and the second part can be viewed as a classifier, f_c, with a trainable parameter, θ_c. Mathematically, the network is expected to solve the following problem:

$$(\theta_e^*, \theta_c^*) \in \arg\min_{\theta_1,\theta_2} \mathcal{L}(f_c \circ f_e(\mathbf{X}), \theta_c, \theta_e, \boldsymbol{y}), \quad (6)$$

where $\mathbf{X}$ and $\boldsymbol{y}$ denote the training input and target, respectively. Suppose we insert an auxiliary network f_1 with parameter θ_1 after f_e and then optimize both the proxy task loss $\mathcal{L}_1(f_1 \circ f_e(\mathbf{X}))$ and $\mathcal{L}(f_c \circ f_e(\mathbf{X}))$, the total classification loss becomes to: $\mathcal{L}_C = \frac{1}{1+\alpha} \cdot (\mathcal{L} + \alpha\mathcal{L}_1)$, where α denotes the weight of proxy task loss. And when N auxiliary modules are added to the network for training, the total classification loss is:

$$\mathcal{L}_C = \frac{1}{1+N\alpha} \cdot (\mathcal{L} + \alpha \sum_i^N \mathcal{L}_i). \quad (7)$$

Under this situation, an auxiliary module is similar to the extra exits used in GoogleNet (Szegedy et al., 2015) to solve the gradient vanishing problem. Because f_1 is not specially designed, and network training necessitates optimizing both $\mathcal{L}_1$ and $\mathcal{L}$, obtaining better performance with an auxiliary module than end-to-end training is difficult.

Surrogate Module. Consider a special case where the structure of the auxiliary module f_1 is exactly the same as the f_c and f_1 does not have the gradient error problem. In this situation, we can simply optimize $\mathcal{L}_1$ and then copy θ_1 to θ_c to train the SNN. When the structure of the auxiliary network and the backbone classifier is inconsistent, that is, $f_1 \neq f_c$, but both inputs are $f_e(\mathbf{X})$ from f_e, and the optimization goals are the same ($\mathcal{L}_1 \rightarrow 0$ and $\mathcal{L} \rightarrow 0$). In this case, the output of f_1 and f_c will become closer and closer through training. Because both f_1 and f_c inputs are equal and their outputs are close, the auxiliary network can be regarded as the surrogate module of the f_c during supervised learning. According to the idea of the transfer adversarial attack, the surrogate module will return a close correct gradient to the f_e (Fang et al., 2022; Gil et al., 2019).

Self-distillation Loss. To achieve the goal of returning more precise gradients to f_e, the output of the surrogate module must be close to the output $\boldsymbol{y}_c$ of f_c. The self-distillation learning that makes the surrogate module's output close to the final output has the potential to realize this target, e.g., BYOT (Zhang et al., 2019).We also let the final SNN output $\boldsymbol{y}_c$ learn from the surrogate module output $\boldsymbol{y}_i$. First and foremost, this is intended to bring the output of the surrogate module and f_c closer together and to speed up SNN training. Second, in the early stages of the training phase, the output performance of the ANN surrogate module may exceed that of the f_c. Even when $\boldsymbol{y}_c$ has a higher accuracy than $\boldsymbol{y}_i$, learning from a poor teacher imparts a label smoothing effect on the SNN, making it more generalizable. Mathmatically, the distillation loss can be described as:

$$\mathcal{L}_{KL_i} = \beta_1 KL(\boldsymbol{y}_c, D(\boldsymbol{y}_i), T_{\text{dis}}) + \beta_2 KL(\boldsymbol{y}_i, D(\boldsymbol{y}_c), T_{\text{dis}}), \quad (8)$$

where $D(\cdot)$ means detach the gradient, KL means Kullback-Leibler divergence loss with its temperature T_{dis}, and $\beta_{1|2} \in \{0, 1\}$ are the parameters to control whether to use the two direction Kullback-Leibler divergence loss.

Total surrogate module Loss. The overall loss function is:

$$\mathcal{L}_{\text{total}} = (1-\lambda)\mathcal{L}_C + \frac{\lambda}{2N}\sum_{i}^{N}\mathcal{L}_{KL_i}, \tag{9}$$

where λ is the weight to balance the classification loss and the distillation loss.

4.3. Surrogate Module Design

When designing the surrogate module, we need to consider the balance between the computational complexity and the structural complexity of the network. On the one hand, when the surrogate module is too simple, it may be unable to accurately reflect the output of the SNN, consequently compromising the network's training impact. On the other hand, when the structure of the surrogate module is excessively complex, the computational complexity will increase, thereby reducing the training efficiency. In light of these concerns, we employ bigger convolution kernels and fewer layers on the surrogate module, therefore decreasing its computational complexity and preserving its expressiveness. In our experiments, we directly use a network with three convolutional layers and two fully connected layers as the surrogate module. The kernel sizes of the convolutional layers are correspondingly 7, 5, and 3. On the first two convolutional layers, feature map dimension reduction is conducted to lower the computational complexity of the network. Before the fully connected layers, an adaptive pooling layer is employed to shrink the feature map to a 2×2 size. Finally, the surrogate module structure that we employ is as follows: kC7s2-kC5s2-kC3s1-AdP2 $\times$ 2-FC512-FCm, where C represents the convolutional layer with k channels, s represents the stride, AdP represents the adaptive pooling layer, and FC denotes the fully connected layer, and m is the number of classifier categories. The activation function employs Leaky-ReLU with a negative slope of 0.01, and the input of the surrogate module is the spike frequency of $f_e(\mathbf{X})$.

4.4. Theoretical Analysis

Previous research, such as BYOT (Zhang et al., 2019), has shown that adding auxiliary networks to ANN improves its generalization capability. The surrogate module can be also seen as finding a more suitable proxy task for SNN to improve its performance. In this part, we show how the surrogate module technique in SNNs can help to alleviate the gradient error problem caused by surrogate gradients. For the sake of simplicity, let us insert only one surrogate module after the encoder part f_e. The $\mathcal{L}_{KL_1}$ converges in the late stages of training, indicating that the surrogate module's output y_1 is close to the SNN output y_c. We can quantify their difference by a small value ε: $\boldsymbol{y}_1 = \boldsymbol{y}_c + \boldsymbol{\epsilon}$.

The gradient can be described as:

$$\frac{\partial \mathcal{L}_1}{\partial \boldsymbol{y}_1} = \boldsymbol{y}_1 - \hat{\boldsymbol{y}} = \boldsymbol{y}_c + \boldsymbol{\epsilon} - \hat{\boldsymbol{y}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_c} + \boldsymbol{\epsilon}, \tag{10}$$

where $\hat{\boldsymbol{y}}$ denotes the one-hot coding of target. Similarly, we can deduce:

$$\frac{\partial \mathcal{L}_{KL_1}}{\partial \boldsymbol{y}_1} = \frac{\partial \mathcal{L}_{KL_1}}{\partial \boldsymbol{y}_c} + \boldsymbol{\epsilon}', \tag{11}$$

where $\boldsymbol{\epsilon}'^k = \boldsymbol{\epsilon}^k - \sum_{j=1}^{J}(\boldsymbol{y}_1^j\boldsymbol{\epsilon}^k + \boldsymbol{y}_c^k\boldsymbol{\epsilon}^j)$ is also very close to zero. Then, we use σ to represent the gradient error accumulation value that result from the surrogate gradient:

$$SG(\frac{\partial \boldsymbol{y}_c}{\partial \theta_e}) = G + \sigma, \tag{12}$$

where SG means calculate the gradient with surrogate gradient and G represents the accurate enough gradients of $\frac{\partial \boldsymbol{y}_c}{\partial \theta_e}$ that can be obtained by FDG (Li et al., 2021) or NA (Yang et al., 2021), though at a huge time cost. Note that, σ is not close to zero and cannot be ignored. When we apply the standard direct training (SDT) with a surrogate gradient, the proportion of the gradient error is $K_{\text{sdt}} = \frac{\sigma}{G}$.

When we implant a surrogate module, the gradient of θ_e changes to:

$$\begin{aligned}\frac{\partial \mathcal{L}_{total}}{\partial \theta_e} = &\frac{1-\lambda}{1+\alpha}\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_c}(G+\sigma+\alpha\cdot S)\\ &+\frac{\lambda}{2}\frac{\partial \mathcal{L}_{KL_1}}{\partial \boldsymbol{y}_c}(G+\sigma+S) + o(\cdot),\end{aligned} \tag{13}$$

where $S = \frac{\partial \boldsymbol{y}_1}{\partial \theta_e}$ represents the gradients through the surrogate module, $o(\cdot)$ is the part that close to zero. In this case, the scale of the gradient error K_{sml} falls into the range $(\frac{\sigma}{G+S}, \frac{\sigma}{G+\alpha\cdot S})$. Compare the gradient error scales by two different training method, $K_{\text{sml}} < K_{\text{sdt}}$ indicates that the surrogate module method can effectively reduce the effect of gradient error on SNN training. Particularly, when $S = G$ and $\alpha = 1$, K_{sml} decreases to about half of K_{sdt}.

Our training pipeline is detailed in Algo. 1. It's worth mentioning that when training is completed, the pipeline will delete all surrogate modules so that surrogate module learning does not affect the SNN inference.

5. Experiments

In order to verify the effectiveness of our method, we provide sufficient experiments on different datasets. The datasets in this section include CIFAR-10, CIFAR-100, ImageNet, and ES-ImageNet. Among them, CIFAR-10, CIFAR-100, and ImageNet are commonly used static image datasets, and ES-ImageNet are neuromorphic datasets created by ImageNet.

Algorithm 1 Training pipeline for surrogate module learning.

```
Input: SNN model $f_{\text{snn}}$, surrogate model channels $k$ and insert position: $p$, hyper-parameter $\lambda$, $\alpha$, Training epochs $E$, training iteration in one epoch: $I_{train}$
for $k_i = 1$ to $k$ do
    Create $f_i$ with $k_i$ channels
    Insert every $f_i$ to $f_{\text{snn}}$ with the position $p_i$
end for
Initialize SNN and all surrogate modules' parameters
initialize the loss function (Eqn. 7, 8, 9) with $\lambda$ and $\alpha$
for $e = 1$ to $E$ do
    for $i_{train} = 1$ to $I_{train}$ do
        Optimizing the loss function:
        $(1-\lambda)\mathcal{L}_C + \frac{\lambda}{2N}\sum_i^N \mathcal{L}_{KL_i}$ (Eqn. 9)
    end for
end for
for $k_i = 1$ to $k$ do
    Delete $f_i$
end for
```

5.1. Implementation Details

We find that it is hard to obtain an optimal learning rate and weight decay with the SGD optimizer when the SNN has multiple exits. As a result, in our experiments, we use the AdamW optimizer with a weight decay of 0.02. The learning rate will cosine decay to 0 during the training for all experiments. The LIF neuron model has the same hyper-parameters as TET (Deng et al., 2021), which means $V_{th} = 1$ and $\tau = 0.5$.

5.2. Ablation Study

In this part, we use the ResNet-18 SNN network with time step $T = 2$ as the basic network. All the experiments are training with 200 epochs and the learning rate is 0.01.

Effect of the surrogate modules on ANN Here, we first validate our method on ANN. Since the gradient on ANN is exact, end-to-end learning is able to acquire well-performed ANN. We need to provide experimental proof that the multiple exit scheme does not affect the final performance too much on ANN. We use the ANN trained with SGD optimizer as our baseline because the performance of ANN trained by AdamW optimizer (learning rate 0.01, weight decay 0.02) is significantly worse than that of ANN trained by SGD optimizer (learning rate 0.1, weight decay $5e-4$). Then, at the 4-th basic block of ResNet-18, we insert one surrogate module ($\lambda = 1/3$, $\alpha = 1$) and train the ANN with the AdamW optimizer at a learning rate of 0.01, weight decay of 0.02. The results were slightly lower than end-to-end training: -0.35% on CIFAR-10 and -0.31% on CIFAR-100. The reason for this phenomenon could be that we are using a different optimizer or that the final network classifier learns from a weaker teacher. Even so, it can be seen that the surrogate module method has the ability to obtain a high-performing network and is ideally suited for implantation into SNN training where gradient error exists.

Table 1. Verify surrogate module method on ANN.

	AdamW	SGD	AdamW Surrogate Module
CIFAR-10	94.73	95.6	95.25
CIFAR-100	71.00	78.45	78.14

Table 2. Accuracy under different learning strategy.

β_{Local}	1				0			
β_1	0	1	0	1	0	1	0	1
β_2	0	0	1	1	0	0	1	1
Accuracy	68.8	69.7	69.7	72.4	70.7	71.8	71.9	74.5

Effect of different self-distillation strategies (β_1, β_2) Here, we adjust β_1 and β_2 to verify the impact of different self-distillation strategies. Meanwhile, we define $\beta_{\text{Local}} \in \{0, 1\}$ to control if we are using the local learning strategy, in which the backbone network's backward gradient is detached at the position of the surrogate module and the preceding encoding part acquires the gradient only from the connected surrogate module. Only one surrogate module is used in this part with $k = 256$, $\lambda = 0.9$ and $\alpha = 0.5$. Combining these three factors (Case $= \{\beta_{\text{Local}}, \beta_1, \beta_2\}$) yields eight situations, and their results are shown in Table 2. When the Case is $\{1, 0, 0\}$, it is common local learning, and its result is only 68.8%. When the Case is $\{1, 1, 1\}$, the accuracy reaches 72.43%, which is 3.63% higher than the case that only uses local learning. This demonstrates that simply applying an auxiliary network does not help SNN training. However, turning it into a surrogate module through self-distillation can effectively facilitate SNN training. The complete surrogate module learning ($\{0, 1, 1\}$) yields the best result, which is 3.79% higher than the standard end-to-end training ($\{0, 0, 0\}$) and 2.59% higher than the BYOT (Zhang et al., 2019) technique ($\{0, 0, 1\}$). And $\{0, 1, 1\}$ is 2.06% higher than $\{1, 1, 1\}$, which indicates that the backward gradient from the backbone network is necessary, even though there is gradient error in it.

Additional training costs for SML Here we provide information on the additional time and memory overhead of using SML (using two surrogate modules) compared to direct training on ResNet18. As shown in Table 4, when $T = 1$, using SML will increase training time by 53.84% and memory usage by 11.34%. However, as simulation time increases, the additional overhead of SML decreases. When

Table 3. Compare with existing works on CIFAR datasets. Our method improves network performance across all tasks. † denotes introduces additional floating-point multiplications, * denotes applying autoaugmentation and cutout.

Dataset	Methods	Type	Architecture	Time Steps	Accuracy (%)
CIFAR-10	IM-Loss (Guo et al., 2022b)	Hybrid	ResNet-19	6	95.49±0.05
				4	95.40±0.08
				2	93.85±0.10
	InfLoR-SNN *† (Guo et al., 2022a)	Surrogate Gradient	ResNet-19	6	96.49±±0.08
				4	96.27±0.07
				2	94.44±0.08
	GLIF† (Yao et al.)	Surrogate Gradient	ResNet-19	6	95.03±0.08
				4	94.85±0.07
				2	94.44±0.10
	our method	Surrogate Module	ResNet-18	6	**95.12**±0.10
				4	**95.01**±0.08
				2	**94.58**±0.18
			ResNet-19	4	**95.54**±0.03
				2	**95.28**±0.15
	our method*	Surrogate Module	ResNet-18	4	**96.04**±0.10
			ResNet-19	4	**96.82**±0.13
CIFAR-100	Dspike (Li et al., 2021)	Surrogate Gradient	ResNet-18	6	74.24±0.10
				4	73.35±0.14
				2	71.68±0.12
	TET (Deng et al., 2021)	Surrogate Gradient	ResNet-19	6	74.72±0.28
				4	74.47±0.15
				2	72.87±0.10
	InfLoR-SNN *† (Guo et al., 2022a)	Surrogate Gradient	ResNet-19	6	79.51±±0.11
				4	78.42±0.09
				2	75.56±0.11
	GLIF† (Yao et al.)	Surrogate Gradient	ResNet-19	6	77.35±0.07
				4	77.05±0.14
				2	75.48±0.08
	our method	Surrogate Module	ResNet-18	6	**78.00**±0.19
				4	**77.36**±0.14
				2	**76.44**±0.15
			ResNet-19	4	**79.18**±0.13
				2	**78.38**±0.30
	our method*	Surrogate Module	ResNet-18	4	**79.49**±0.11
			ResNet-19	4	**81.70**±0.17

Table 4. Additional training costs for SML on ResNet18.

Time Step	T=1	T=2	T=3	T=4	T=5
Time Cost (%)	+53.84	+27.42	+20.86	+11.88	+11.04
Memory Cost (%)	+11.34	+8.22	+8.56	+6.13	+2.44

we use the commonly used simulation time of 4 in SNN, the additional time overhead is only 11.88% and the additional memory overhead is only 6.13%. We believe that this level of additional training overhead is acceptable when compared to the significant improvement in SNN performance brought by SML.

5.3. Comparison to exiting works

In this section, we compare our experimental results with current SOTA work. Note that, to improve the SNN performance, several current works introduce some additional floating-point operations in the SNN; for instance, GLIF (Yao et al.) uses some learnable gating units to control the leaky and reset of the LIF model, and InfLoR-SNN (Guo et al., 2022a) applies a non-linear function to membrane potential before the step function. On the contrary, our method will delete the surrogate modules during the network inference, retaining just the standard SNN backbone without introducing any additional floating-point operations, which is advantageous for embedding in neuromorphic hardware.

CIFAR. For the static CIFAR-10 and CIFAR-100 datasets, we validate our method on both commonly used network architectures, ResNet-18 and ResNet-19. It should be noted that ResNet-19 (2.22 GFlops), which is significantly larger than ResNet-18 (0.56 GFlops), is a network architecture designed specifically for SNN (Zheng et al., 2021) on CIFAR. It expands the space domain to alleviate the absence of a binary spike. We employ two surrogate modules ($N = 2$), which are positioned after the third and sixth basic modules ($p = 3, 6$), respectively. The surrogate modules have 256 channels ($k = 256$). And we set the $\lambda = 0.9$ and $\alpha = 1/2$. Our training pipeline has 300 epochs with learning rate 0.01 and batchsize 256. As shown in Table 3, compare to these works (IM-Loss and TET) that dose not introduce additional floating-point multiplications, our method enhance the SNN accuracy both on CIFAR-10 (0.05% to IM-Loss)

Table 5. Compare with existing works on large-scale datasets. Our method improves network performance across all tasks. † denotes introduces additional floating-point multiplications.

Dataset	Methods	Type	Architecture	Time Steps	Accuracy (%)
ImageNet	tdBN (Zheng et al., 2021)	Surrogate Gradient	ResNet-34	6	63.72
	SEW † (Fang et al., 2021)	Surrogate Gradient	SEW-ResNet-34	4	67.04
	Tandem Learning (Wu et al., 2021b)	Tandem Learning	VGG-16	16	65.08
	TET (Deng et al., 2021)	Surrogate Gradient	ResNet-34	6	64.79
	TEBN (Duan et al.)	Surrogate Gradient	ResNet-34	4	64.29
	IM-Loss (Guo et al., 2022b)	Hybrid	ResNet-34	6	67.43±0.11
	GLIF† (Yao et al.)	Surrogate Gradient	ResNet-34	4	67.52
	our method	Surrogate Module	ResNet-18	4	64.53
				2	62.49
			ResNet-34	6	**69.35**
				4	**68.25**
				2	**65.77**
ES-ImageNet	ES-ImageNet (Lin et al., 2021)	Surrogate Gradient	ResNet-18	8	39.89
			ResNet-34	8	43.42
	Bridge Conversion (Lin et al., 2022)	Hybrid	ResNet-18	8	43.74
	ConvECLIF2D-A † (Wu et al., 2022)	Surrogate Gradient	ResNet-18	8	44.25
	our method	Surrogate Module	ResNet-18	8	**44.76**

and CIFAR-100 (4.46% to TET). We also verify our method with stronger data augmentation (Guo et al., 2022a), as a result, the SNN achieves a performance of 81.86% at T=4 on CIFAR-100 datasets and ResNet-19 structure, which is 2.19% higher than the current SOTA results.

ImageNet. ImageNet (Deng et al., 2009) is the most popular large-scale dataset that contains more than 1280k training images and 50k validation images. We also use the standard augmentation pipeline and crop the images to 224×224. Following tdBN (Zheng et al., 2021), we remove the original Maxpooling and setting the first basic block stride to 2 for downsampling. For surrogate module learning, we use 2 surrogate modules for ResNet-18 and 3 surrogate modules for ResNet-34, and we place them on the positions $p = 3, 6$ and $p = 3, 7, 13$, respectively. And we haven't searched for the optimal hyperparameters of loss Eqn. 9, but just set $k = 256$, $\lambda = 1/3$ and $\alpha = 1$. We adopt an AdamW optimizer with a learning rate 4e-3, which is cosine decay to 0, and the weight decay is 0.02. We use the TIT method to reduce training time (Deng et al., 2021; Hu et al., 2021). First, we train the SNN for 160 epochs with $T = 2$ (batch size is 512); Then we change the time steps T to 4, learn rate 4e-4, batch size 256, and continue training for another 80 epochs; Finally, we continue changing the time steps T to 6 and training for another 40 epochs. The results are summarized in Table. 5. Under the same architecture, our method enhances the SNN accuracy by 1.92% compared to the hybrid training method IM-Loss and significantly improves the accuracy by 3.46% compared to the training from scratch method. In comparison to other SNN-based works but introduce the floating-point multiplications, our ResNet-34 result is marginally better than SEW's (Fang et al., 2021)and GLIF (Yao et al.).

ES-ImageNet. ES-Imagenet (Lin et al., 2021) is one of the most challenging event-stream datasets. It is derived from ImageNet and contains 1257K training images and 50K test images. We use the same pre-process pipeline as in previous work (Lin et al., 2021) and the same hyperparameter settings as in the ImageNet experiments, with the exception of adjusting the learning rate to 4e-4 and batch size to 192. We train the dataset with 50 epochs from scratch without any other technique. During the training phase, we discovered that it is very easy to overfit on this massive, noisy dataset. When the accuracy rate of the training set reaches to 64.7%, the accuracy of the test set is only 44.76%. As a result, the accumulate gradient error may not be the most important factor affecting test accuracy. However, when compared to previous work that used the same LIF model and ResNet-18 structure, we still have a significant improvement in accuracy $(+4.87\%)$.

6. Conclusion

This paper focuses on the gradient error accumulation problem in SNN training and proposes surrogate module learning to mitigate its effects by creating a new path to backpropagate the gradient. On almost all mainstream datasets, we significantly improve SNN performance.

References

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pp. 583–593. PMLR, 2019.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Decoupled greedy learning of cnns. In *International Conference on Machine Learning*, pp. 736–745. PMLR, 2020.

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

DeBole, M. V., Taba, B., Amir, A., Akopyan, F., Andreopoulos, A., Risk, W. P., Kusnitz, J., Otero, C. O., Nayak, T. K., Appuswamy, R., et al. Truenorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5):20–29, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., Zhao, G., Li, P., and Xie, Y. Rethinking the performance comparison between snns and anns. *Neural Networks*, 121: 294 – 307, 2020.

Deng, S., Li, Y., Zhang, S., and Gu, S. Temporal efficient training of spiking neural network via gradient reweighting. In *International Conference on Learning Representations*, 2021.

Duan, C., Ding, J., Chen, S., Yu, Z., and Huang, T. Temporal effective batch normalization in spiking neural networks. In *Advances in Neural Information Processing Systems*.

Duan, S. and Principe, J. C. Training deep architectures without end-to-end backpropagation: A brief survey. *arXiv preprint arXiv:2101.03419*, 2021.

Fang, S., Li, J., Lin, X., and Ji, R. Learning to learn transferable attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 571–579, 2022.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. Deep residual learning in spiking neural networks. *arXiv preprint arXiv:2102.04159*, 2021.

Friston, K. Learning and inference in the brain. *Neural Networks*, 16(9):1325–1352, 2003.

Gil, Y., Chai, Y., Gorodissky, O., and Berant, J. White-to-black: Efficient distillation of black-box adversarial attacks. *arXiv preprint arXiv:1904.02405*, 2019.

Guo, Y., Chen, Y., Zhang, L., Wang, Y., Liu, X., Tong, X., Ou, Y., Huang, X., and Ma, Z. Reducing information loss for spiking neural networks. In *European Conference on Computer Vision*, pp. 36–52. Springer, 2022a.

Guo, Y., Tong, X., Chen, Y., Zhang, L., Liu, X., Ma, Z., and Huang, X. Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 326–335, 2022b.

Herranz-Celotti, L. and Rouat, J. Surrogate gradients design. *arXiv preprint arXiv:2202.00282*, 2022.

Hu, Y., Wu, Y., Deng, L., and Li, G. Advancing residual learning towards powerful deep spiking neural networks. *arXiv preprint arXiv:2112.08954*, 2021.

Kaiser, J., Mostafa, H., and Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

Kim, S., Park, S., Na, B., and Yoon, S. Spiking-yolo: Spiking neural network for energy-efficient object detection. *arXiv preprint arXiv:1903.06530*, 2019.

Lee, J. H., Delbruck, T., and Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., and Gu, S. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Li, Y., Kim, Y., Park, H., Geller, T., and Panda, P. Neuromorphic data augmentation for training spiking neural networks. *arXiv preprint arXiv:2203.06145*, 2022.

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

Lin, Y., Ding, W., Qiang, S., Deng, L., and Li, G. Es-imagenet: A million event-stream classification dataset for spiking neural networks. *Frontiers in neuroscience*, pp. 1546, 2021.

Lin, Y., Hu, Y., Ma, S., Yu, D., and Li, G. Rethinking pre-training as a bridge from anns to snns. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

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

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., and Luo, Z.-Q. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12444–12453, 2022.

Pyeon, M., Moon, J., Hahn, T., and Kim, G. Sedona: Search for decoupled neural networks toward greedy block-wise learning. In *International Conference on Learning Representations*, 2020.

Roy, K., Jaiswal, A., and Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Shrestha, S. B. and Orchard, G. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, pp. 1412–1421, 2018.

Suetake, K., Ikegawa, S.-i., Saiin, R., and Sawada, Y. S^2 nn: Time step reduction of spiking surrogate gradients for training energy efficient single-step neural networks. *arXiv preprint arXiv:2201.10879*, 2022.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021a.

Wu, J., Xu, C., Han, X., Zhou, D., Zhang, M., Li, H., and Tan, K. C. Progressive tandem learning for pattern recognition with deep spiking neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021b.

Wu, Y., Deng, L., Li, G., Zhu, J., and Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12: 331, 2018.

Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., and Shi, L. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1311–1318, 2019.

Wu, Z., Zhang, Z., Gao, H., Qin, J., Zhao, R., Zhao, G., and Li, G. Modeling learnable electrical synapse for high precision spatio-temporal recognition. *Neural Networks*, 149:184–194, 2022.

Xiao, M., Meng, Q., Zhang, Z., Wang, Y., and Lin, Z. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. *Advances in Neural Information Processing Systems*, 34:14516–14528, 2021.

Yang, Y., Zhang, W., and Li, P. Backpropagated neighborhood aggregation for accurate training of spiking neural networks. In *International Conference on Machine Learning*, pp. 11852–11862. PMLR, 2021.

Yao, M., Gao, H., Zhao, G., Wang, D., Lin, Y., Yang, Z., and Li, G. Temporal-wise attention spiking neural networks for event streams classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10221–10230, 2021.

Yao, X., Li, F., Mo, Z., and Cheng, J. Glif: A unified gated leaky integrate-and-fire neuron for spiking neural networks. In *Advances in Neural Information Processing Systems*.

Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., and Ma, K. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3713–3722, 2019.

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11062–11070, 2021.

Zhu, R.-J., Zhao, Q., Zhang, T., Deng, H., Duan, Y., Zhang, M., and Deng, L.-J. Tcja-snn: Temporal-channel joint attention for spiking neural networks. *arXiv preprint arXiv:2206.10177*, 2022.

A. Effect of the surrogate modules number (N)

We experiment with the number of surrogate modules since it influences the training effect and training costs. We utilize 1, 2, and 3 surrogate modules and evenly insert them into the ResNet-18. We placed the surrogate module after the fourth basic block if using only one surrogate module. In cases 2 and 3, two surrogate modules are placed after the third and sixth basic blocks, and three surrogate modules are placed after the second, fourth, and sixth basic blocks, respectively. We set the $k = 256$, $\alpha = 1$, and $\lambda = 1/3$. The results are shown in Table. 6. Even in the situation of simply setting hyperparameters, it is evident that increasing the number of surrogate modules can improve the training impact of the SNN. However, increasing the number of surrogate modules will increase the training time. For instance, when the number of surrogate modules is 1 or 2, the training time rises by 13.3% and 25.5%, and the SNN performance increases by 2.45% and 3.19%, respectively. The network performance only improves by 0.29% when the number of surrogate modules increases from 2 to 3. Considering the increase in training time, the improvement in SNN performance is not obvious, hence we employ 2 surrogate modules on ResNet-18.

Table 6. Accuracy and time cost under the different numbers of surrogate modules.

	$N=0$	$N=1$	$N=2$	$N=3$
Accuracy	69.65	72.10	72.84	73.13
Training Time (s)	6043	6868	7583	9959

B. Effect of the channels of the surrogate module (k)

In this section, we use two surrogate modules with $\lambda = 0.9$ and $\alpha = 0.5$ (placed after the third and sixth basic modules) to test training performance under different surrogate module channel numbers k. The number of channels will affect the computational cost and surrogate modules' expressiveness. The outcomes are presented in Tabel 7. We found that when k is too small, such as 64, the surrogate module usually cannot adequately reflect the SNN's output, resulting in a significant accuracy drop. When k is too large, such as $k = 512$, the computational complexity of the surrogate module will increase. And the final accuracy, in this case, is slightly worse than in the situation where $k = 256$. This might be because the network has not been sufficiently trained when the surrogate module is complex. As a result, we recommend setting the $k = 256$ for the surrogate module.

Table 7. Accuracy under different channel number k of surrogate module.

k	64	128	256	512
Accuracy	73.64	74.14	74.49	74.31
Training Time (s)	6570	6870	7583	9400

C. Effect of surrogate module position.

Here, we also set $k = 256$, $\alpha = 1$, and $\lambda = 1/3$, and only consider the influence of surrogate module position when the number of surrogate modules is one. In the three cases, the surrogate module is positioned in the second, fourth, and sixth places of the basic block sequence, respectively. The results are displayed in Table 8. It can be seen that the placement of the surrogate module has minimal influence on the final performance. This is because of the balance between the subsequent classification block's surrogate gradient error accumulation and the learning capacity. Some local learning studies have also discovered the second term: the optimal local learning position is more likely to be found in the shallow layer rather than the deep layer (Pyeon et al., 2020). In this paper, we always utilize the surrogate module positions to uniformly split all basic block sequences on the ResNet structure network.

Table 8. Accuracy under different placement of surrogate modules.

Position	$p=2$	$p=4$	$p=6$
Accuracy	72.05	72.10	72.16

D. Greedy search for optimal hyper-parameters on CIFAR-100

Different datasets may have varying optimum hyper-parameters for surrogate module learning; however, in this part, we simply employ a greedy search to identify the ideal hyper-parameters for CIFAR-100.

λ denotes the weight of distillation loss. Here, we analyze the performance of ResNet18 on CIFAR100 under various commonly-used λ values. We use two surrogate modules ($N = 2$), placed after the 3rd and 6-th basic blocks, respectively, and keep $k = 256$ and $\alpha = 1.0$. As shown in Table 9, SNN's performance will increase proportionally as λ gradually increases. When λ is 0.9, the SNN performance is at its peak. In addition, we examined the performance of the network when lambda is 0.99. In this situation, the proportion of distillation loss is significantly more than that of classification loss, but the network's accuracy is still very high (73.97).

Table 9. Accuracy under different λ.

λ	0	$1/5$	$1/3$	$1/2$	$2/3$	$3/4$	$9/10$	$99/100$
Accuracy	70.69	72.64	72.84	73.03	73.82	73.97	74.10	73.97

Then we set λ to its optimal value of 0.9, and search for the ideal value of α. α represents the weight of the classification loss of the surrogate modules. The results (Table 10) show that the value α does not significantly affect the final SNN performance, and its optimal value on CIFAR100 is 0.5.

Table 10. Accuracy with different α.

α	$1/4$	$2/4$	$3/4$	1
Accuracy	74.31	74.49	73.97	74.10

After determining the optimal $lambda$ and $alpha$, we begin our search for the T_dis. The T_{dis} denotes the temperature of the distillation loss. As shown in Table 11, the optimal temperature is 3. As a result, both too strict and too loose distillations are detrimental to SNN surrogate gradient learning. It is worth noting that the optimal T_dis value is also the optimal value in the CIFAR-DVS dataset. And the temperature value is the same as in previous work (Zhang et al., 2019).

Table 11. Accuracy under different distillation temperature T_{dis}.

T_{dis}	1	2	3	4	5
Accuracy	72.80	74.2	74.49	73.87	73.60

E. SML adaptability to long simulation time.

As gradient errors caused by surrogate gradients also accumulate over time, here we verified the performance of SML over long simulation times. We used one surrogete module, placed after the 4th basic block of ResNet18, with hyperparameters α set to 0.5 and λ set to 0.9. The results are summarized in Table 12. Direct training achieved the highest accuracy at T=6. However, as T increased to 10, the accuracy decreased by 0.32% despite SNN expression enhancement. On the other hand, SML achieved the best accuracy at T=8, and when T increased to 10, the accuracy only dropped by 0.02%. These results suggest that SML can also reduce the accumulation of gradient errors over time to some extent.

F. Accuracy on CIFAR10-DVS.

On the CIFAR10-DVS dataset, we set the hyper-parameters $\lambda = 1/3$, $\alpha = 1$, and $k = 256$. We employ two surrogate modules, which are put after the third and sixth ResNet-18 basic blocks, respectively. The results are summarized in Table. 13. Our stated result (only surrogate module) is slightly inferior to the TCJA-TET-SNN (-0.11%) However, with the aid of TET loss, we may obtain an 85.23% SOTA accuracy.

G. Changes in accuracy during training

As demonstrated in Fig. 3, the accuracy of the training set improves more rapidly with surrogate module learning than with standard direct training, indicating that surrogate module may give more accurate gradients for SNN's f_e. At the same time,

Table 12. Accuracy comparison under different simulation time.

T	4	6	8	10
SDT	71.95	72.54	72.29	72.22
SML	76.62	77.08	77.22	77.20

Table 13. Compare with existing works on CIFAR10-DVS datasets. Our method improves network performance across all tasks. † denotes introduces additional floating-point multiplications.

Dataset	**Methods**	**Type**	**Architecture**	**Time Steps**	**Accuracy (%)**
DVS-CIFAR10	Dspike (Li et al., 2021)	Surrogate Gradient	ResNet-18	10	75.4±0.05
	NDA (Li et al., 2022)	Surrogate Gradient	VGG-11	10	81.7
	InfLoR-SNN† (Guo et al., 2022a)	Surrogate Gradient	ResNet-19	10	75.50±0.12
	TET (Deng et al., 2021)	Surrogate Gradient	VGGSNN	10	83.17±0.15
	TCJA-TET-SNN† (Zhu et al., 2022)	Surrogate Gradient	VGGSNN	10	83.3
	Our method (+TET)	Surrogate Module	VGGSNN	10	84.60
	Our method	Surrogate Module	ResNet-18	10	83.19 ±0.41
	Our method (+TET)	Surrogate Module	ResNet-18	10	**85.23** ± 0.52

surrogate module learning requires approximately half as many training epochs as stardand direct training the same SNN performance.

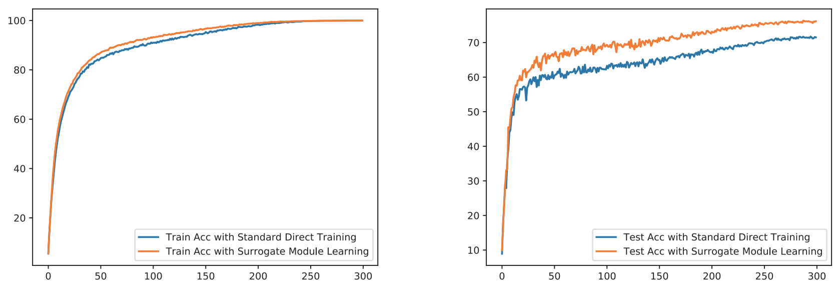

Figure 3. Comparison of the training speed of standard direct training and surrogate module training on CIFAR-100.