EFFICIENT SURROGATE GRADIENTS FOR TRAINING SPIKING NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking Neural Network (SNN) is widely regarded as one of the next-generation neural network infrastructures, yet it suffers from an inherent non-differentiable problem that makes the traditional backpropagation (BP) method infeasible. Surrogate gradients (SG), which are an approximation to the shape of the Dirac's δ-function, can help alleviate this issue to some extent. To our knowledge, the majority of research, however, keep a fixed surrogate gradient for all layers, ignorant of the fact that there exists a trade-off between the approximation to the delta function and the effective domain of gradients under the given dataset, hence limiting the efficiency of surrogate gradients and impairing the overall model performance. To guide the shape optimization in applying surrogate gradients for training SNN, we propose an indicator χ, which represents the proportion of parameters with non-zero gradients in backpropagation. Further we present a novel χ-based training pipeline that adaptively makes trade-offs between the surrogate gradients' shapes and its effective domain, followed by a series of ablation experiments for verification. Our algorithm achieves 69.09% accuracy on the ImageNet dataset using SEW-ResNet34 - a 2.05% absolute improvement from baseline. Moreover, our method only requires extremely low external cost and can be simply integrated into the existing training procedure.

1 INTRODUCTION

Spike Neural Networks (SNN) have gained increasing attention in recent years due to their biological rationale and potential energy efficiency as compared to the common real-value based Artificial Neural Networks (ANN). SNN communicates across layers by the addition of spiking signals. On the one hand, this spiking mechanism turns multiplicative operations to additive operations, increasing the inference procedure's efficiency. On the other hand, it introduces an intrinsic issue of differentiability, which makes training SNNs more challenging. At present, the method for obtaining practical SNNs can be roughly divided into three categories: converting a pretrained ANN to SNN (Sengupta et al., 2019; Deng & Gu, 2020; Li et al., 2021a; Bu et al., 2021), training with biological heuristics methods (Hao et al., 2020; Shrestha et al., 2017; Lee et al., 2018), and training with BP-like methods (Wu et al., 2018; Zheng et al., 2020; Li et al., 2021b; Yang et al., 2021). The converting method may not promote increased inference efficiency in practice since it requires a lengthy simulation period (high inference latency) to catch up to the accuracy of the source ANN (Sengupta et al., 2019; Rueckauer et al., 2017). Although the biological heuristics technique requires just local information to change network parameters, it is confined to small datasets due to its limitation in representing global information (Wu et al., 2018; Shrestha et al., 2017). Compared to these two approaches, direct training with BP-like method is capable of handling complex models with a very short simulation duration to attain adequate model performance (Zheng et al., 2020; Fang et al., 2021; Li et al., 2021b).

With the help of surrogate gradient, the SNN can be directly trained through the BPTT algorithm on an ANN-based platform (Wu et al., 2018). Nonetheless, there is a non-negligible performance disparity between directly trained SNN and ANN, particularly on large and complicated datasets (Deng et al., 2020; Jin et al., 2018). This is because training SNN with surrogate gradient can only obtain approximate gradients, and the final performance is highly affected by the surrogate gradient shape. A more suitable surrogate gradient shape usually results in a better performing SNN (Neftci et al., 2019). However, an appropriate surrogate gradient must strike a compromise between the approximation shape and the effective domain of gradients. So just altering the shape of the

surrogate gradient to be more similar to the δ-function may result in the training failing due to gradient disappearance, as the gradients of most membrane potentials are extremely small. Additionally, the optimal surrogate gradient shapes for various layers may different and may change throughout the training process (Li et al., 2021b). As a result, using a fixed initial surrogate gradient shape (adequate effective domain) during the whole training phase will always have a substantial gradient error, which affects the final training result.

The purpose of this work is to optimize the SNN training pipeline by adaptively altering the shape of surrogate gradient in order to control the effective domain for the surrogate gradients. We suggest an index χ to denote the proportion of membrane potential with non-zero gradients in backpropagation and present a technique to control the proportion of non-zero gradients (CPNG) in the network. The CPNG technique modifies the shape of surrogate gradients during network training, progressively approaching the δ-function while maintaining the index χ steady within an effective range to ensure training stability. Finally, each layer succeeds in finding a surrogate gradient shape that makes a better balance between the approximation error to the δ-function with the size of effective domain than the fixed-shape surrogate gradients. It's worth mentioning that our strategy only incurs minor additional costs during the training phase and has no effect on the inference phase. We verify the compatibility of CPNG to the existing mainstream SNN infrastructures such as VGG (Simonyan & Zisserman, 2014), ResNet (He et al., 2016), and Sew-ResNet (Fang et al., 2021). In all reported comparative experiments, training with CPNG gives more accurate models than training with vanilla surrogate gradients.

Our main contributions can be summarized as follows:

- We identify and investigate the impact of the shape of surrogate gradients on SNN training. Our finding characterizes a special representative power for SNN that can be utilized to improve its performance.
- We propose a statistical indicator χ for the domain efficiency of surrogate gradients and a χ-based training method CPNG that adjusts the shape of surrogate gradients through the training process, driving the surrogate gradients close to the theoretical δ-function with ensured trainability on sufficiently large domains.
- Our CPNG method improves classification accuracy on both static image datasets including CIFAR10, CIFAR100 and ImageNet, as well as event-based image datasets such as CIFAR10-DVS. We achieve an accuracy of 69.09% in the experiment that trains ImageNet on Sew-ResNet34.

2 Related Work

There are two primary branches of training a high-performing deep spiking neural network, converting a pretrained artificial neural network to its corresponding spiking neural network, and directly training a spiking neural network through BP-like method.

ANN-SNN Conversion ANN-SNN conversion takes advantage of the high performance of ANN and converts the source ANN to the target SNN through weight-normalization (Diehl et al., 2015; 2016) or threshold balancing (Sengupta et al., 2019). However, SNN forming this method requires a huge simulation length to catch up with the source ANN's performance. Numerous strategies have been proposed to shorten the simulation time, including robust threshold (Rueckauer et al., 2016), SPIKE-NORM (Sengupta et al., 2019), and RMP (Han et al., 2020). A work (Deng & Gu, 2020) examines the conversion error theoretically, decomposes it layer by layer, and offers threshold ReLU and shift bias procedures to decrease the error. Based on it, Li et al. (Li et al., 2021a) divide the conversion error into clip error and floor error and design adaptive threshold, bias correction, potential correction, and weight calibration to dramatically decrease the required simulation length. A recent work (Bu et al., 2021) further proposes unevenness error, trains ANN with a novel activation function and reduce simulation length.

BP-like Method HM2-BP (Jin et al., 2018) enables SNN to adjust the spike sequence rather than just the spike at a certain moment. TSSL-BP (Zhang & Li, 2020) decomposes the backpropagation error into inter and intra interactions, calculating the derivatives only at the spiking moment. NA algorithm (Yang et al., 2021), which calculates the gradient of the non-differentiable part through

finite difference. The surrogate gradient based BP algorithm uses differentiable functions instead of the Dirac's δ-function to mitigate the non-differentiation problem in the process of SNN training. Different from ANNs, SNNs naturally have time attribute. Therefore, the existing studies consider both temporal and spatial information in BP procedure (Wu et al., 2018). Normalization methods are crucial in SNN training, which help speed up the network's convergence and prevent gradient disappearance or explosion. For this purpose, SNN-friendly normalization algorithms such as NeuNorm (Wu et al., 2019) and threshold-dependent batch normalization (tdBN) (Zheng et al., 2020) have been developed. The majority of studies have employed fixed shape surrogate gradients, and some work has preliminarily explored the performance of surrogate gradients with different shapes (Wu et al., 2018; Neftci et al., 2019). A recent work looked into the shape-changing surrogate gradient (Li et al., 2021b), proposed Dspike as surrogate gradient, and used finite difference to guide the change of Dspike's shape, significantly improving the performance of SNN. In order to train a very deeper SNN, Sew-ResNet (Fang et al., 2021) structure is proposed, which enables SNN training even on a 152-layer network.

3 PRELIMINARY

Through out the paper, we use bold letters to denote matrices and vectors, superscripts to identify specific layers, subscripts to denote specific neurons, and indexes to identify specific moments.

Leaky Integrate-and-Fire Model. We use the Leaky Integrate-and-Fire (LIF) module for spiking neurons. Formally, given the pre-synaptic input (denoted by $c_i^{(l)}[t+1]$) of the ith neuron in the lth layer at time step$t+1$, we can model the iterative process in LIF as

$$c_i^{(l)}[t+1] = \sum_{j}^{N^{(l-1)}} w_{ij}^{(l)} s_j^{(l-1)}[t+1], \tag{1}$$

$$u_i^{(l)}[t+1] = \tau u_i^{(l)}[t](1 - s_i^{(l)}[t]) + c_i^{(l)}[t+1], \quad s_i^{(l)}[t+1] = H(u_i^{(l)}[t+1] - \nu). \tag{2}$$

Here, $N^{(l-1)}$ is the number of neurons in the $(l-1)^{th}$ layer, $s_j^{(l-1)}[t+1]$ is the output spike of the jth neuron in the $(l-1)^{th}$ layer at time $t+1$, $w_{ij}^{(l)}$ is the weight between j^{th} neuron in $(l-1)^{th}$ layer and i^{th} neuron in l^{th} layer, $u_i^{(l)}[t]$ is the membrane potential of the ith neuron in the lth layer at time t, τ is the membrane potential attenuation factor, $H(\cdot)$ is the step function, and ν is the activation threshold. When the membrane potential of a neuron exceeds the activation threshold, a spike is released and the membrane potential of the current neuron is set to zero.

Surrogate Gradient Function. There are various surrogate gradient shapes adopted by previous work (Wu et al., 2018; Neftci et al., 2019). In this work, we used triangle-like function, rectangular-like function and arctan-like function to verify the effectiveness of CPNG. These functions are described below:

$$f_{\text{triangle}}(x) = \begin{cases} \beta(1 - \beta|x-\nu|) & \text{if} \quad |x-\nu| < 1/\beta \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

$$f_{\text{rectangular}}(x) = \begin{cases} \beta & \text{if} \quad |x-\nu| < 1/(2\beta) \\ 0 & \text{otherwise} \end{cases}, \quad f_{\text{arctan}}(x) = \frac{\beta}{1 + (\pi\beta(x-\nu))^2}, \tag{4}$$

where β represents the maximum gradient value of current surrogate gradient function. Notably, the surrogate gradient satisfy $\int_{-\infty}^{+\infty} f(x) = 1$, which is also the property of the δ-function.

Loss Function. In our experiments, we use cross-entropy-loss ($\mathcal{L}_{CE}$) as the loss function. The formula is given as

$$\text{out} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{W}^{(L)} \cdot \mathbf{S}^{(L-1)}[t], \quad \text{Loss} = \mathcal{L}_{CE}(\text{out}, \text{label}), \tag{5}$$

where T represents the simulation time, L represents the last layer of the SNN, $\mathbf{W}^{(L)}$ represents the weight matrix of the Lth layer, and $\mathbf{S}^{(L-1)}$ represents the output spike vector of the $(L-1)^{th}$ layer. Consistent with (Wu et al., 2019; He et al., 2016), our final output layer is a voting layer devoid of any LIF model.

4 METHOD

4.1 SHAPE PARAMETERS AND EFFECTIVE DOMAIN INDICATOR

Shape Parameters. Intuitively, increasing the shape parameter β of surrogate gradients would drive it closer to the δ-function (Fig. 1(right)). One might expect to adopt a very high β to obtain SNN for a good performance. We first examine whether this intuitive approach is possible. We trained VGG16-structured SNN (replace ReLU with LIF, and switch from maxpooling to avgpooling.) on CIFAR100 using triangle-like surrogate gradient with β set from 0.25 to 2 respectively. As shown in Fig. 1(left), the test accuracy increases when β varies from 0.25 to 1.0 but remains at 1.0% when β is set to 1.5 and 2.0, indicating that properly increasing β may benefit the training but arbitrarily increasing β will drive the training collapse. According to another viewpoint (Zenke & Vogels, 2021; Herranz-Celotti & Rouat, 2022), when the beta is greater than 1, the gradient explosion on the deep SNN will occur, resulting in training failure; thus, some of the most recent SG fixes the maximum value of SG at 1 (Suetake et al., 2022; Zenke & Ganguli, 2018). But the narrow effective interval of the SG is the primary culprit in our experiments. We detected the maximum gradient absolute values for SNN training at the first 100 batches when $\beta = 1(1.0001)$ and $\beta = 1.5(1.000)$, respectively. These results demonstrate that when β is 1.5, SNNs with the VGG16 architecture and a timestep of 5 are insufficiently deep to have gradient explosions.

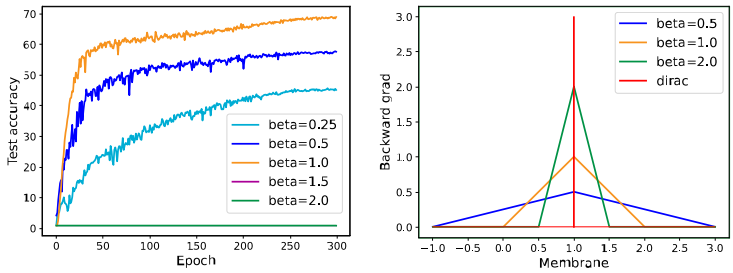

Figure 1: Left: Test accuracy of different β when threshold is 1.0. Right: δ-function and triangle-like surrogate gradient when the threshold is 1.0.

In fact, these results unveil that efficient training of SNNs requires not only the approximation to the δ-function but also the insurance for the surrogate gradients to work. Thus it is necessary to employ a dynamic shape-changing strategy rather than using a fixed-shape surrogate gradient. This issue is also covered by (Li et al., 2021b) as well and was owed to the lack of adaption to the dataset variation.

Effective Domain Indicator. As aforementioned, if we unrestrictedly increase the β in order to get the surrogate gradient closer to the δ-function, the training curve will remain flat without any update. This is caused by the fact that the majority of the membrane potential remain outside the effective region with zero gradients as it gets overly small as β increases. On the other hand, if β is too small, it will also lead to a suboptimal training outcome due to the presence of a substantial gradient error (Fig. 1(left)) between the adopted surrogate gradients and the δ-function. Thus a *proper* surrogate gradients should maintain an optimal balance between the domain effectiveness and the δ-function approximation.

To quantitatively guide the choice of β, we need a statistical indicator to denote the percentage of membrane potentials that fall into the domain of surrogate gradients. As illustrated in Fig. 2(left), the distribution of membrane potentials on each layer takes the normal shape. Thus, for the simplicity of calculation, we regard the membrane potential distribution of all neurons within the same layer as a Gaussian one. By calculating the mean μ and the standard deviation σ of the membrane potential

before this layer releases spikes, we can obtain the proportion of the neuron with a non-zero gradient during a certain iteration (the area between the red lines in Fig. 2(right)). For a given β, the effective gradient domain of triangle-like surrogate gradient is $[\nu - 1/\beta, \nu + 1/\beta]$, we can obtain the definite integral of the current normal distribution in this effective gradient domain, which is the *Effective Domain Indicator* χ:

$$\chi = \int_{\nu-1/\beta}^{\nu+1/\beta} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx. \tag{6}$$

Based on this indicator χ, we can then effectively determine to what extent we can tune the β while ensuring that there are enough membrane potentials located within the effective range of surrogate gradients to make the training progress.

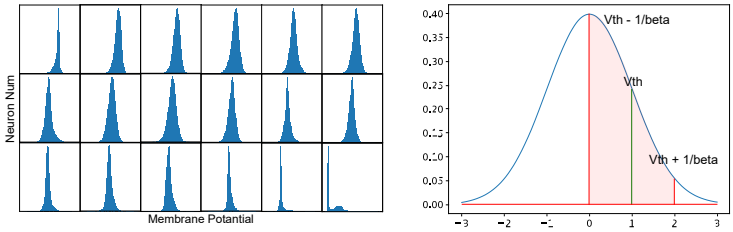

Figure 2: Left: Membrane distribution of each layer in experiment training ResNet19 on CIFAR10. Right: When the threshold is 1.0 and β is 1.0, the proportion of neurons with non-zero gradient.

4.2 CPNG METHOD

In this section, we will cover how to combiningly optimize β and χ to maximize the effectiveness of surrogate gradients. To train the network successfully, there must be sufficient membrane potential values in the effective domain of the surrogate gradient, i.e., χ must be large enough. The most extreme case is $\beta \to 0$, which gives $\chi \to 1$. Obviously, this is not an optimal solution as it introduces substantial error for the gradients. In order to determine the choice of β for a given χ, we need to investigate the gradient form $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{S}^{(l)}[t]}$ to see how the β affects the error back-propagation.

When $l = L-1$, we can directively get $\frac{\partial \mathcal{L}_{CE}^{(1)}}{\partial \mathbf{S}^{(L-1)}[t]}, t = 1, 2, \cdots, T$ according to Eqn. 5.

When $l = 0, 1, \cdots, L-2$, we can first derive $\frac{\partial u_i^{(l)}[t+1]}{\partial s_i^{(l)}[t]} = -\tau u_i^{(l)}[t]$ and $\frac{\partial u_i^{(l)}[t+1]}{\partial u_i^{(l)}[t]} = \tau(1 - s_i^{(l)}[t])$ from Eqn. 2, and further have

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{u}^{(l)}[t]} = \begin{cases} \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}^{(l)}[t]} \frac{\partial \mathbf{s}^{(l)}[t]}{\partial \mathbf{u}^{(l)}[t]} & t = T \\ \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}^{(l)}[t]} \frac{\partial \mathbf{s}^{(l)}[t]}{\partial \mathbf{u}^{(l)}[t]} + \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{u}^{(l)}[t+1]} \cdot \tau(1 - \mathbf{s}^{l}[t]) & t = 1, 2, \cdots, T-1 \end{cases}, \tag{7}$$

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}^{(l)}[t]} = \begin{cases} \frac{\partial \mathcal{L}_{CE}^{(1)}}{\partial \mathbf{s}^{(l)}[t]} & t = T \\ \frac{\partial \mathcal{L}_{CE}^{(1)}}{\partial \mathbf{s}^{(l)}[t]} + \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{u}^{(l)}[t+1]} \cdot (-\tau \mathbf{u}^{l}[t]) & t = 1, 2, \cdots, T-1 \end{cases}. \tag{8}$$

by Eqn. 2. Here, $\mathbf{u}^{(l)}[t]$ represents the membrane potentials vector of the lth layer, $\frac{\partial \mathcal{L}_{CE}^{(1)}}{\partial \mathbf{s}^{(l)}[t]}$ is the gradient directly obtained from previous layer. The complete $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}^{(l)}[t]}$ needs to consider the dependence between the current moment spike and the membrane potential at the next moment (Eqn. 8). The term $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{u}^{(l)}[t+1]}$ can be obtained iteratively from the gradients from its succeeding layer, and $\frac{\partial \mathbf{s}^{(l)}[t]}{\partial \mathbf{u}^{(l)}[t]}$ is given by surrogate gradients. We can conclude that each item of the $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{u}^{(l)}[t]}$ contains $\frac{\partial \mathbf{s}^{(l)}[t]}{\partial \mathbf{u}^{(l)}[t]}$, while effective interval of surrogate gradient (χ) determines $\frac{\partial \mathbf{s}^{(l)}[t]}{\partial \mathbf{u}^{(l)}[t]}$, which in turn affects the calculation of $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial \mathcal{L}_{CE}^{(1)}}{\partial \mathbf{s}^{(l-1)}[t]}$. However, the surrogate gradient is only an approximation of δ-function, and

Algorithm 1 Control the Proportion of Non-zero Gradient

Input: SNN model with L layer, current iterator epoch e,
χ_{limit}, and vector χ_{recorder}: store each layer's smallest χ
Output: Each layer's surrogate gradient parameter β
if e == 0 **then**
  for $l = 1, 2, \cdots L$ **do**
    calculate current χ by Eqn.6 for layer-l and store at $\chi_{\text{recorder}}[l]$
  end for
else
  for $l = 1, 2, \cdots L$ **do**
    calculate current χ_{cur} by Eqn.6 for layer-l
    if $\chi_{\text{recorder}}[l] < \chi_{\text{limit}}$ **then**
      $\chi_{\text{recorder}}[l] = \chi_{\text{limit}}$
    else if $\chi_{\text{cur}} < \chi_{\text{recorder}}[l]$ **then**
      $\chi_{\text{recorder}}[l] = \chi_{\text{cur}}$
    end if
    $\chi_{\text{min}} = \chi_{\text{recorder}}[l]$
    if $\chi_{\text{min}} \neq \chi_{\text{cur}}$ **then**
      use χ_{min} to update β using binary search method.
    end if
  end for
end if
return β for each layer

an overly relaxed surrogate is harmful to the network's final performance. If we reasonably restrict the effective interval of the surrogate gradients, it is possible to drive the SNN to select those more essential membrane potentials for backpropagation.

We also need to ensure that the new χ does not make the network difficult to train, for this, CPNG sets the target χ of each layer to the smallest χ that has occurred in the current layer during the training iteration, rather than an artificial goal. If the network can be trained when the smallest χ appears, then the network should still be trained after we adjust the β and maintain the smallest χ. When using CPNG, we expect that the network parameters are appropriate, that is, the network has traversed the whole dataset to prevent the misleading of network parameters by data randomness. For example, if we use CPNG once per batch, the network parameters are mostly affected by the first few batches in the early stages of network training, and the statistical indicator χ obtained by using such network parameters will have a lot of randomnesses.

CPNG computes the smallest χ of each layer during the iteration process as χ_{min} and records it. If the χ value of a certain layer rises after an epoch, CPNG adjusts the χ value of the current layer to χ_{min} by increasing the β, otherwise, keep the current β fixed and update χ_{min}. Since the χ_{min} of each layer of neurons may be different, different layers may have different surrogate gradient shapes. In addition, we set a safe lower bound χ_{limit}. When χ falls below the lower bound, the β may decrease to force χ back above χ_{limit} in order to guarantee that sufficient membrane potential values are covered in the effective domain of surrogate gradient for successful SNN training. In fact, we test the performance of χ_{limit}=0.05 and χ_{limit}=0.2 on VGG16+CIFAR100 experiment and there is only a small gap between them. In our experiments, we set χ_{limit}=0.2 uniformly. The CPNG algorithm is detailed in Algo.1.

4.3 The Cost of CPNG Method

The extra cost of CPNG occurs in two steps: (1) collecting the mean and variance of the membrane potential before releasing the spikes of each layer; (2) altering the β using the indicator χ. In our experiment, we only use the mean and variance of the last batch to calculate the indicator χ, which makes the cost of the first step in the same order of magnitude as the batch normalization (Ioffe & Szegedy, 2015) operation. For the latter step, we provide a binary search method that solves the problem very fast, and further optimization algorithms can further improve the solution speed. The above analysis is the cost of using CPNG once, the overall cost takes into account the frequency of using CPNG. In our experiments, we employ CPNG just once every epoch, which is quite economical

Table 1: The impact of using time and batch size on Vgg16+CIFAR100 experiment.

	256(w/o CPNG)	256(B)	256(E)	512(w/o CPNG)	512(B)	512(E)
Acc	69.08%	71.05%	71.54%	68.44%	69.59%	70.20%
Time	705.59m	799.41m	716.67m	377.76m	424.49m	381.79m

B: Use CPNG per batch. E: Use CPNG per epoch (last batch)

when compared to the network training time. Quantitatively, in the VGG16+CIFAR100 experiment, it takes an average of 1.7GFLOPs to obtain the output corresponding to an input without CPNG, and using CPNG will only add 5.59×10^{-3} additional GFLOPs. Using CPNG for each epoch's training time only requires an extra 3.30 seconds in average (1.57% of total training time).

5 Experiment

To verify the effectiveness of the CPNG method, we provide groups of comparative experiments (Sec. 5.2) on both static and neuromorphic datasets. We also compare CPNG with existing works in Sec. 5.3 and show the final β of SNN after training with CPNG in Sec. 5.4.

5.1 Implementation Details

All the SNN architectures include the tdBN layer (Zheng et al., 2020) with the average-pooling layer, and compared to their ANN versions, we replace activation function ReLU with LIF. Our experiment settings, such as optimizer, learning rate, are detailed in Appendix. A.2. Except for applying the CPNG method at the end of each epoch, all other settings, such as learning rate, batch size, etc., are consistent. The data preprocessing for each dataset included in the experiments are as follows:

CIFAR10 and ImageNet. We use standard processing methods for these datasets, see appendix. A.1 for more details.

CIFAR10-DVS. CIFAR10-DVS is an event-based image datasets. Following the previous work (Li et al., 2021b), we re-sample 10 simulation length, divide the dataset into 9k training images and 1k test images, reduce the spatial resolution to 48×48, and apply data augmentation techniques.

Table 2: χ_{limit} Experiments on ResNet18+CIFAR-DVS.

χ_{limit}	0.05	0.1	0.2	0.3	0.4	0.5	0.6
Accuracy	76.6%	76.6%	76.37%	76.8%	76.0%	76.1%	74.1%

5.2 Performance improvement with CPNG

Although χ is employed as a statistical indicator, table1 demonstrates that increasing the batch size does not always result in better performance of CPNG. This is due to the fact that the performance of the neural network is not linear with the batch size. We don't need to increase batch size excessively to increase the sample size of χ. Besides, as we mentioned in 4.2, using CPNG once per batch does not guarantee performance improvement, it may bring more overhead and worse performance. Then we compared the impact of different χ_{limit} on CPNG, table2 shows that we may don't need to focus too much on χ_{limit}, $0.05 - 0.2$ might be suitable.

We also verified whether CPNG can be applied to various surrogate gradient functions. For infinite non-zero interval surrogate gradients, such as arctan, we directly use $[\nu - 1/\beta, \nu + 1/\beta]$ as the integration interval to calculate the indicator χ, then use Algo.1 to solve new β. Considering the case of using triangle-like surrogate gradient, we control the percentage of membrane potentials with non-zero gradients, which can also be explained as: under the current surrogate gradient, find a representative interval related to β, and make the calculated χ in this interval equal to the target value. In the case of using surrogate gradients with infinite non-zero gradient interval, the representative interval for calculating the indicator is $[\nu - 1/\beta, \nu + 1/\beta]$, if we need a smaller interval $[\nu - r_1, \nu + r_1]$

Table 3: Examine CPNG on various SGs.

Dataset	Method	Architecture	Time Step	Accuracy
CIFAR10-DVS	Triangular	ResNet18	10	75.6%
	Triangular+CPNG	ResNet18	10	76.37±1.02 %
	Rectangular	ResNet18	10	74.7%
	Rectangular+CPNG	ResNet18	10	75.3%
	ArcTan	ResNet18	10	67.2%
	ArcTan+CPNG	ResNet18	10	67.5%
	Triangular+TET	ResNet18	10	79.9%
	Triangular+TET+CPNG	ResNet18	10	**82.3%**

Table 4: Result of training spiking neural network.

Dataset	Method	Architecture	Time Step	Accuracy
CIFAR10	STBP-tdBN (Zheng et al., 2020)	ResNet19	6	93.16%
	ANN-to-SNN (Li et al., 2021a)	ResNet20	32 64 128	94.78% 95.30% 95.42%
	ANN-to-SNN (Bu et al., 2021)	ResNet20	8 16 32	89.55% 91.62% 92.24%
	CPNG	ResNet19	6	**94.02**±0.05**%**
CIFAR100	Diet-SNN (Rathi & Roy, 2020)	VGG16	5	69.67%
	ANN-to-SNN (Li et al., 2021a)	VGG16	32 64 128	73.55% 76.64% 77.40%
	ANN-to-SNN (Bu et al., 2021)	VGG16	8 16 32	73.96% 76.24% 77.01%
	CPNG	VGG16	5	**71.32**±0.20**%**
CIFAR10-DVS	Dspike (Li et al., 2021b)	ResNet18	10	75.40%
	TET(Deng et al., 2022)	VGG11	10	83.17%
	CPNG	ResNet18	10	76.37%±0.26
	CPNG + TET	VGG11	10	**83.83**±0.02**%**
ImageNet	ANN-to-SNN (Li et al., 2021a)	ResNet34	32 64 128	64.54% 71.12% 73.45%
	ANN-to-SNN (Bu et al., 2021)	ResNet34	16 32 64	59.35% 69.37% 72.35%
	Sew-ResNet (Fang et al., 2021)	Sew-ResNet34	4	67.04%
	TET (Deng et al., 2022)	Spiking-Sew-ResNet34	6	64.79%
	TET (Deng et al., 2022)	Sew-ResNet34	4	68.00%
	CPNG	Sew-ResNet34	4	**69.09%**

with the current mean and standard deviation, just set the new β to $1/r_1$. The experimental results in table3 also demonstrate the effectiveness of our approach (More results are shown in Appendix A.5).

Finally, we verify the compatibility of CPNG with existing direct training methods TET (Deng et al., 2022)(table3), we got a 2.4% improvement on the ResNet18 structure.

5.3 Comparison to Existing Works

In this section, the experimental results we report all use triangle-like surrogate gradient. On some datasets, the current SOTA conversion method performs better than direct training, but they need lengthy simulation time steps, especially on the ImageNet dataset. In addition, the conversion-based method cannot be applied to neuromorphic datasets such as CIFAR10-DVS. Hence, it's also necessary to design an SG search method to optimize the direct training.

VGG16 + CIFAR100. We train the SNN for 300 epochs on 1 TITAN, with the batch size of 256 and the time step of 5. As shown in Table .4, CPNG manages to improve 2.24% accuracy. To show that CPNG effectively control the rise of χ, we report the χ during training in Fig. 3.

ResNet19 + CIFAR10. We use ResNet19 (Zheng et al., 2020) in accordance with tdBN layers, the batch size of 256 and the time step of 6. Then we train the SNN for 200 epochs on 4 TITAN, CPNG achieves an increase in test accuracy to 94.02%, while its counterpart is 93.26%.

ResNet18 + CIFAR10-DVS. We train the model for 200 epochs on 2 TITAN Xp cards, with the time step of 10 and the batch size of 72. The CPNG obtains a higher test accuracy of 76.37% while consuming significantly less overhead than Dspike (Li et al., 2021b). In addition, the combination of CPNG and TET can achieve an average test accuracy of 83.83%, which is a new SOTA.

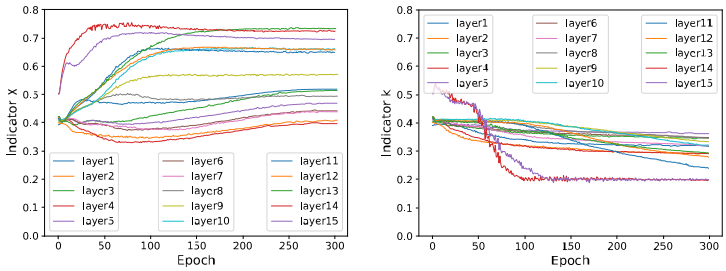

Figure 3: In the experiment of VGG16, the proportion of non-zero gradient membrane potentials of neurons in different layers without CPNG (left) and with CPNG (right).

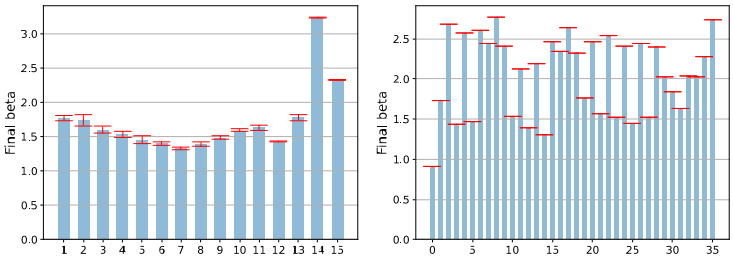

Figure 4: Left: Final β of each layer in VGG16 experiment, the initial β is 1.0. Right: Final β of each layer in Sew-ResNet34 experiment, the initial β is 1.0..

Sew-ResNet34 + ImageNet. We use the same structure and membrane decay rate, etc. as Sew-ResNet (Fang et al., 2021). We train the model for 160 epochs on 8 GTX 3090 cards with the time step of 4 and the batch size of 544. Using only CPNG can achieve an accuracy of 69.04%, surpassing the current direct training SOTA.

5.4 Surrogate Gradient Shapes of Different Layers

We show the final β of some experiments, and all experimental results are presented in the appendix A.3. Various layers' β are different, which demonstrates that various layers match distinct surrogate gradient shape as a result of their varying membrane potential distributions.

CPNG eventually increases the β of most layers. Compare to CPNG, randomly increasing the β can make the network difficult to train (Fig. 1(left)). Even with β set to 1.5, which most of the neuron layers shown in Fig. 4(left) can approach or reach, the network is still difficult to train. This demonstrate that it is safe to increase β using CPNG, while it is unsafe to increase β arbitrarily.

6 Conclusion

This work proposes a new perspective for directing the shape change of the surrogate gradient, we propose a statistical indicator χ that guides the shape change of the surrogate gradient, and propose the CPNG method for modifying the shape of the surrogate gradient during training while guaranteeing the proportion of membrane potential with non-zero gradients. It's possible that the failure to produce satisfactory results when pulling surrogate gradient directly to δ-function is due to a failure to meet the premise that the network can be trained normally. In other words, there may exists a trade-off between the approximation to the δ-function and the effective domain of gradients under the given dataset, and CPNG helps us approach the equilibrium point.

REFERENCES

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 *International Conference on Learning Representations*, 2021.

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.

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2020.

Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. *arXiv preprint arXiv:2202.11946*, 2022.

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, and Shih Chii Liu. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, 2015.

Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8. IEEE, 2016.

Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *arXiv preprint arXiv:2102.04159*, 2021.

Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13558–13567, 2020.

Yunzhe Hao, Xuhui Huang, Meng Dong, and Bo Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Networks*, 121:387–395, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

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

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. *arXiv preprint arXiv:1805.07866*, 2018.

Chankyu Lee, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in neuroscience*, 12:435, 2018.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. *arXiv preprint arXiv:2106.06984*, 2021a.

Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 2021b.

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.

Nitin Rathi and Kaushik Roy. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, 2020.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv preprint arXiv:1612.04052*, 2016.

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.

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.

Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 international joint conference on neural networks (IJCNN)*, pp. 1999–2006. IEEE, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

Kazuma Suetake, Shin-ichi Ikegawa, Ryuji Saiin, and Yoshihide Sawada. S´2 nn: Time step reduction of spiking surrogate gradients for training energy efficient single-step neural networks. *arXiv preprint arXiv:2201.10879*, 2022.

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.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1311–1318, 2019.

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

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

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.

Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *arXiv preprint arXiv:2002.10085*, 2020.

Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. *arXiv preprint arXiv:2011.05280*, 2020.

A Appendix

A.1 Data Processing

CIFAR. For the training set, we randomly crop the image to (32,32) and apply a random horizontal flip. Then all the images will be normalized to a standard normal distribution.

ImageNet. For the training set, we randomly crop the image to (224,224) and apply a random horizontal flip. Then for the test set, we resize the image to (256,256) and center crop the image to (224,224). Finally, all the images are normalized to a standard normal distribution.

A.2 More Experiment Details

In this section, we provide more experimental details. We use 1/20 of the training epochs for warm-up, linearly increase learning rate from $0.1lr$ to lr, then use cosine decay to reduce learning rate to 0 in the remaining epochs. In the last batch, the mean and variance of the membrane potential are additionally stored in the forward process, and after the backward process, the stored mean and variance are used to obtain a new β according to Algorithm 1. Hyperparameters are shown in the table 5.

Table 5: Experiment Setting

Experiment	CIFAR100	CIFAR10	CIFAR-DVS	ImageNet	CPNG + TET
learning rate	0.1	0.1	0.01	0.01	0.001
weight decay	1e-4	1e-4	4e-5	4e-5	4e-5
momentum	0.9	0.9	0.9	0.9	—-
optimizer	sgd	sgd	sgd	sgd	adam
warm-up	True	True	False	False	False

A.3 Experiment Result

In this section, we display the change of indicator χ(Fig. 5, Fig. 6), the final β and the test accuracy of each experiment. Our codes can be found in the supplemental. As shown by Fig. 7, Fig. 8, Fig. 9 and Fig. 10, except for the first layer of the Sew-ResNet34+ImageNet experiment, almost all neuron layers have obtained a steeper surrogate gradient (a larger β), which illustrates that surrogate gradient has further optimization space in the SNN training process. However, as we mentioned in Sec. 4.1, choosing a surrogate gradient closer to the $\delta-$function at beginning will make training the network difficult, therefore we'll need tools (such as CPNG) to assist us in finding a better surrogate gradient during training. In the Sew-ResNet34+ImageNet experiment, CPNG finds a smaller β for the first layer to ensure the proportion of membrane potential with non-zero gradients. This suggests that when backpropagating with surrogate gradient in deeper networks, the initial few layers may only have a small proportion of their parameters updated, and we should account for this more.

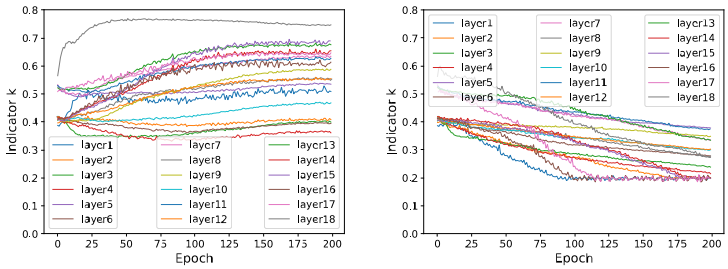

Figure 5: In the experiment of ResNet19, the proportion of non-zero gradient membrane potentials of neurons in different layers without CPNG (left) and with CPNG (right).

A.4 Membrane Potential Distribution

In this section, we show the membrane potential distribution of each layer of neurons in all experiments(Fig. 11, Fig. 12, Fig. 13). The layer after BN is more consistent with the normal distribution than the layer after FC. But for the convenience of calculation, we approximate the membrane potential distribution of all neurons to the normal distribution.

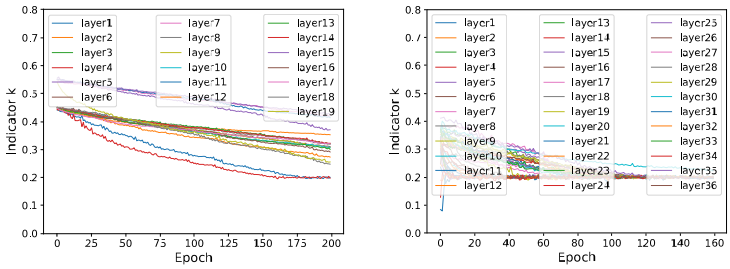

Figure 6: The proportion of non-zero gradient membrane potentials of neurons in different layers with CPNG in the experiment of ResNet18 (left) and Sew-ResNet34 (right).

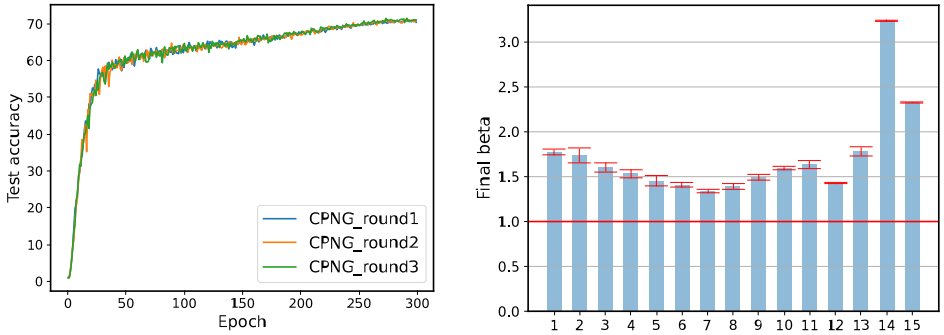

Figure 7: Left: Test accuracy of triplicate VGG16+CIFAR100 experiments using CPNG. Right: Final β of each layer in CIFAR100 experiment, the initial β is 1.0

A.5 Adjust Other Surrogate Gradients

The results we report are based on triangle-like surrogate gradient, a surrogate gradient with finite non-zero interval, but this does not mean that our method can only be used for finite non-zero interval surrogate gradients. All results are shown in Tab. 6.

A.6 Saliency Map

The gradient noise generated by surrogate gradient may affect the network's attention location, resulting in a more blurred saliency map. And there's no doubt that a more appropriate surrogate gradient will produce fewer gradient noises. Thus, the saliency map's clarity may be utilized to determine the fitness of surrogate gradient. We used SmoothGrad (Smilkov et al., 2017), which reduces the effect of visually noise. The results of CIFAR100+VGG16 experiment are shown in Fig. 14. We discovered that CPNG can assist the model to identify clearer contour information. For example, in the boys category, the saliency map obtained by using CPNG is more accurately positioned on the person's face, whereas traditional surrogate gradient also pays much attention to the surrounding background; in the elephants category, the use of CPNG can clearly see the trunk as well as the elephant's outline, whereas traditional surrogate gradient can only obtain blurred borders.

A.7 Loss Landscape

In order to illustrate the trainability of the model after using CPNG, we use the loss landscape to show the change of loss after changing the model parameters. If the loss landscape does not reveal more non-convex regions than the traditional surrogate gradient, it means that using CPNG will not make convergence more difficult. If the same weight offset is applied for various networks when displaying

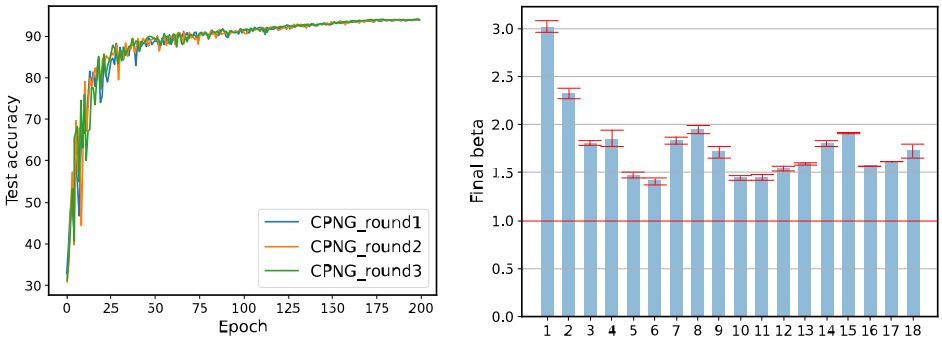

Figure 8: Left: Test accuracy of triplicate ResNet19+CIFAR10 experiments using CPNG. Right: Final β of each layer in CIFAR10 experiment, the initial β is 1.0

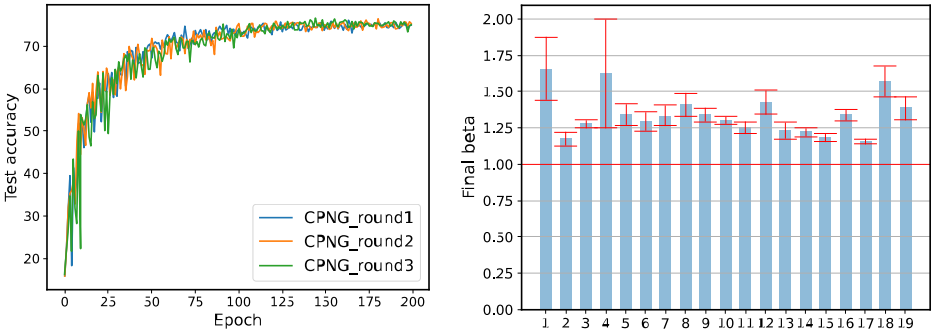

Figure 9: Left: Test accuracy of triplicate ResNet18+CIFAR10-DVS experiments using CPNG. Right: Final β of each layer in CIFAR10-DVS experiment, the initial β is 1.0

the loss landscape, the networks with bigger weight will exhibit more stationarity. However, due to the existence of batch normalization, the weight scaling of the network has no effect on the inference results, so the sharpness of loss landscape of different networks may only be due to the weight scaling. To explain the model performance between the traditional surrogate gradient and CPNG, we use the loss landscape demonstration with filter-wise normalization (Li et al., 2017) that mitigates the effect of weight scaling and correlates the model's generalization ability to the flatness of the loss landscape. As shown in Fig. 15, in the experiment of ResNet19+CIFAR10, utilizing CPNG can obtain a smoother minimum and a wider locally convex region, implying that CPNG has no negative effect on network's convergence difficulty.

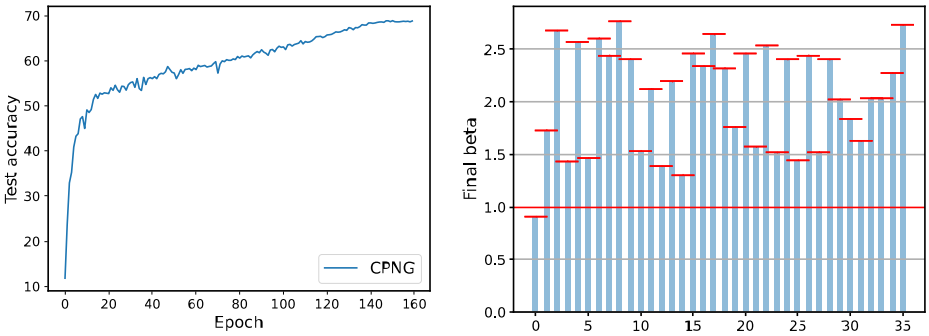

Figure 10: Left: Test accuracy of Sew-ResNet34+ImageNet experiments using CPNG. Right: Final β of each layer in ImageNet experiment, the initial β is 1.0

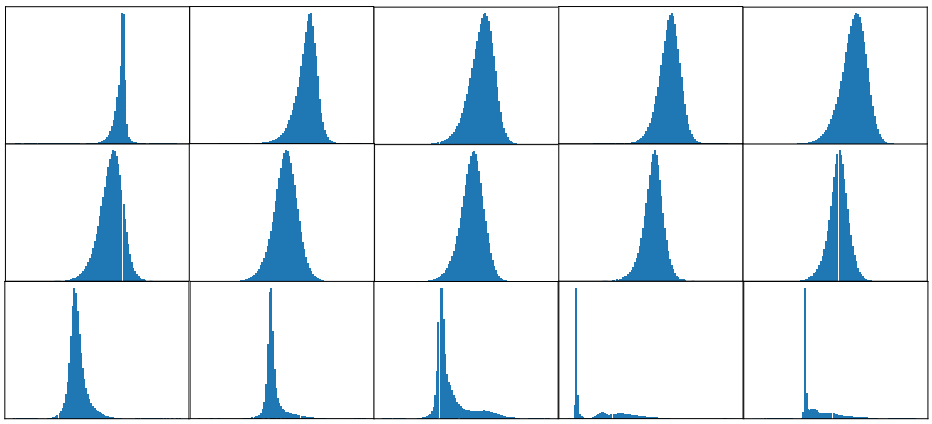

Figure 11: Membrane distribution of each layer in experiment training VGG16 on CIFAR100

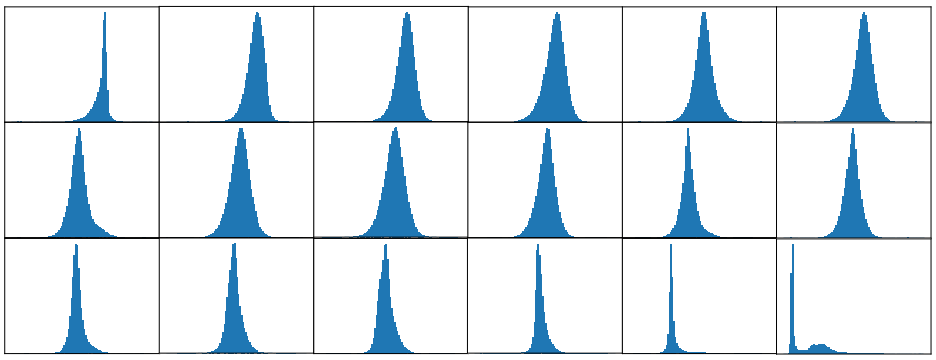

Figure 12: Membrane distribution of each layer in experiment training ResNet19 on CIFAR10

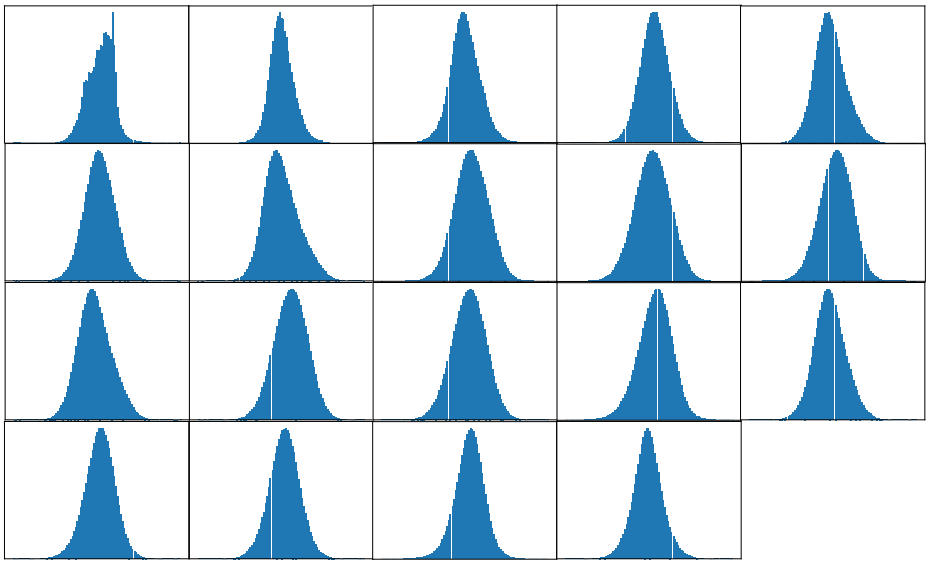

Figure 13: Membrane distribution of each layer in experiment training ResNet18 on CIFAR-DVS

Table 6: Examine CPNG on various SGs.

Dataset	Method	Architecture	Time Step	Accuracy
CIFAR10	Triangular	ResNet19	6	93.26%
	Triangular+CPNG	ResNet19	6	94.02±0.05%
	Rectangular	ResNet19	6	90.5%
	Rectangular+CPNG	ResNet19	6	93.07±0.04%
	ArcTan	ResNet19	6	93.24%
	ArcTan+CPNG	ResNet19	6	93.95%
CIFAR100	Triangular	VGG16	5	69.08%
	Triangular+CPNG	VGG16	5	71.32±0.20%
	Rectangular	VGG16	5	65.54%
	Rectangular+CPNG	VGG16	5	67.74±1.02%
	ArcTan	VGG16	5	68.15%
	ArcTan+CPNG	VGG16	5	69.49±0.20%
CIFAR10-DVS	ArcTan	ResNet18	10	67.2%
	ArcTan+CPNG	ResNet18	10	67.5%
	Triangular+TET	ResNet18	10	79.9%
	Triangular+TET+CPNG	ResNet18	10	82.3%
	Triangular	ResNet18	10	75.6%
	Triangular+CPNG	ResNet18	10	76.37±1.02 %

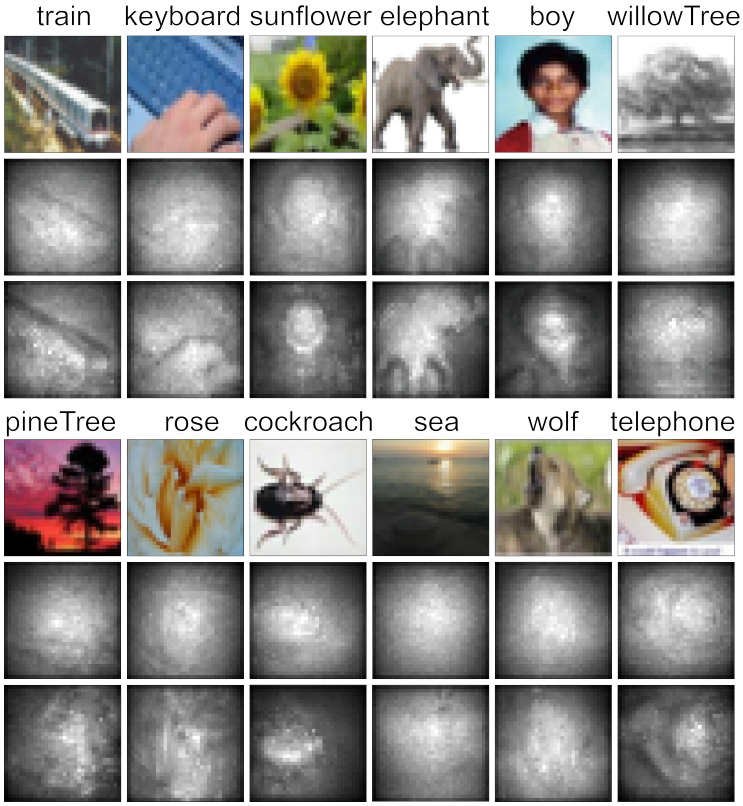

Figure 14: Saliency map. Three photos are a group, the top is the original image, the middle is the salilency map given by the model trained by traditional surrogate gradient, and the bottom is the saliency map given by the model trained with CPNG.

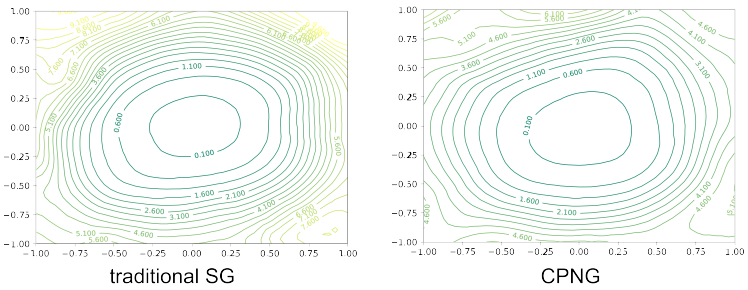

Figure 15: Loss landscape of traditional surrogate gradient and CPNG.