SSF: Accelerating Training of Spiking Neural Networks with Stabilized Spiking Flow

Jingtao Wang[1,4,5], Zengjie Song[2], Yuxi Wang[3],
Jun Xiao[1*], Yuran Yang[6], Shuqi Mei[6], Zhaoxiang Zhang[1,3,4,5*]
[1]University of Chinese Academy of Sciences [2]Xi'an Jiaotong University
[3]Centre for Artificial Intelligence and Robotics, HKISI-CAS
[4]Institute of Automation, Chinese Academy of Sciences
[5]State Key Laboratory of Multimodal Artificial Intelligence Systems [6]Tencent
{wangjingtao2021, zhaoxiang.zhang}@ia.ac.cn, zjsong@hotmail.com, xiaojun@ucas.ac.cn

Abstract

Surrogate gradient (SG) is one of the most effective approaches for training spiking neural networks (SNNs). While assisting SNNs to achieve classification performance comparable to artificial neural networks, SG suffers from the problem of time-consuming training, preventing it from efficient learning. In this paper, we formally analyze the backward process of classic SG and find that the membrane accumulation through time leads to exponential growth of training time. With this discovery, we propose Stabilized Spiking Flow (SSF), a simple yet effective approach to accelerate training of SG-based SNNs. For each spiking neuron, SSF averages its input and output activations over time to yield stabilized input and output, respectively. Then, instead of back propagating all errors that are related to current neuron and inherently entangled in time domain, the auxiliary gradient is directly propagated from the stabilized output to input through a devised relationship mapping. Additionally, SSF method is suitable to different neuron models. Extensive experiments on both static and neuromorphic datasets demonstrate that SNNs trained with SSF approach can achieve performance comparable to the original counterparts, while reducing the training time significantly. In particular, SSF speeds up the training process of state-of-the-art SNN models up to 10× when time steps equal to 80.

1. Introduction

Benefiting from huge amount of data and advances in computing hardwares, deep artificial neural networks (ANNs) have achieved rapid development in past decades

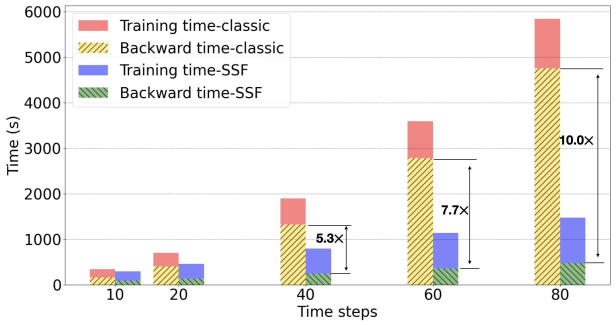


Figure 1. Training time and backward time cost in one epoch with different time steps. Classic SG method [7] experiences an exponential growth of training time when number of time steps increases. By contrast, SSF method significantly decreases training time cost, especially when training SNNs with long time steps.

and even outperform human beings in areas like image classification [14], semantic segmentation [38] and object detection [25]. However, the promising performance of ANNs builds on the expense of substantial computational resources and power consumption during both training and inference, limiting its application in edge devices.

Known as the third generation of neural networks, Spiking Neural Network (SNN) simulates biological neuron's action potential process which accumulates inputs as membrane potential and generates a binary spike when exceeding threshold. This simulation brings SNN with special neural dynamics, strong biological plausibility, and promising high energy efficiency on neuromorphic hardwares [26]. These advantages make SNN potential in solving many problems ANN encountered, including limited energy supplies in edge devices mentioned before. Hence, SNN has drawn a lot of attention of researchers in recent years.

Yet still training SNN is a big challenge since the discrete nature of binary spikes hampers the effective usage of gradient-based backpropagation (BP) methods on the training of SNN [31]. Among all the studies, one effective idea

*Corresponding authors.

is using some auxiliary variable approximating the non-differentiable part during back propagation. Based on this idea comes surrogate gradient (SG) method, which trains SNNs in the same way of training recurrent neural networks (RNNs) with the help of continuous relaxation of the real gradients [36]. With several years' development, SG method enables SNNs to obtain high accuracy and be promoted to very deep neural networks [7]. However, the computation cost for SG is unacceptably high. As illustrated in Fig. 1, when SNNs' time steps grow longer, time needed in training process increases exponentially. This creates an awkward situation that bigger number of time steps usually leads to better performance as longer spike trains can represent more complex information [7, 12, 41] (shown in Fig. 4). Therefore, it is desired to speed up the training of SG-based SNNs, especially with long time steps, while maintaining competitive performance.

In this paper we propose an acceleration method, dubbed Stabilized Spiking Flow (SSF), for efficiently training SNNs. We first formalize the backward process of classic SG method and identify that the main contributor to the exponential growth of the training time is SNNs' membrane accumulation process which has to calculate gradients sequentially to back-propagation in time domain. To decrease the temporal correlation between time steps, SSF approximates output and input spike trains in each layer with stabilized inputs and outputs. Through the built mapping between stabilized inputs and outputs, SSF skips the membrane accumulation process and thus greatly speeds up the training process. As shown in Fig. 1, the effect of our method becomes more pronounced when time steps grow longer. Our main contributions are summarized as follows:

1. We systematically study the backward process of classic SG method and find that the exponential growth of training cost is mainly caused by SNNs' membrane accumulation process.

2. To deal with the problem, we propose SSF method. SSF approximates each layers' input and output spike trains with stabilized spiking flows and skips the cumbersome backpropagation in time domain by building relationship between inputs and outputs directly.

3. The sufficient experiments on both static and neuromorphic datasets prove the effectiveness of SSF. Particularly, SSF can speed up the backward process of state-of-the-art models up to $10\times$ while achieving nearly the same classification accuracy.

2. Related Work

2.1. Learning Strategy of SNN

Learning strategy plays a very important role in the development of SNN. In general, training methods till now can be divided into three parts: conversion methods, bio-inspired methods, and adjusted BP methods.

The most representative conversion method is ANN-to-SNN [13, 21, 2, 39, 28, 17, 3]. Typically, It transmits a well-trained ANN into a SNN and then uses some tuning approaches narrowing the gap between them. ANN-to-SNN enables SNNs to perform as well as ANNs. However, this method aims at making a SNN act like an ANN and dismisses the rich time dynamic of SNN. Aside from low performance on neuromorphic datasets, obtained SNN still needs enough time steps to achieve good performance.

STDP method dominates the development of bio-inspired methods in recent years [16, 15, 40, 8, 32, 23, 37, 29]. STDP method trains SNNs directly with Spike-Timing-Dependent Plasticity (STDP) rules. These rules are oriented from Hebb rules,which have been proven plausible learning rules in brain. STDP method has the potential to realize strong artificial intelligence and has already achieved some success in unsupervised learning area [29], but those rules it uses adjust weights with local spike histories rather than global losses, stopping STDP from going deep or being applied to supervised learning. Till now, training SNNs with STDP still cannot obtain satisfying performance.

In the aspect of adjusted BP methods [34, 22, 41, 35, 24, 42, 4, 10], surrogate gradient (SG) method attracts widespread attention recently. SG replaces SNNs' non-differentiable parts with surrogate gradients, thus solving the biggest barrier hindering the usage of classic BP on training SNNs. After several years of development, SG method now can train SNNs with both low latency and competitive performance. Our work focuses on SG method.

2.2. SG Method

SG method uses surrogate gradients replacing the non-differentiable parts during back-propagation and trains SNN like RNN. It was first used by Bothe et al. in 2002 [1], who bypassed the discontinuity problem by linearly processing the relationship between the input received by the neuron and the resulting pulse firing time. In 2016, Lee et al. [19] obtained a SNN with several hidden layers which outperformed ANN on neuromorphic datasets. In 2019, Wu et al. [36] created the STBP framework and got high accuracy on CIFAR-10 dataset. Recently, Guo et al. [12] designed the MPD approach, achieving nearly the same classify accuracy of ANN on CIFAR-10 dataset. In general, SG method can now train SNNs with both high accuracy and low latency, but the time cost is unacceptably high, especially in training SNNs with long time steps.

2.3. Speed Up Approaches

Quite a few works focus on dealing with the high training time cost of SG method. Wu et al. [35] attempted to use a SNN to inference but back-propagate with an ANN, ASF-BP method [34] resembles it while in a different spiking neuron model. These two works all built surrogate gradients between inputs and outputs directly, decreasing training time cost to a very low level. However, they both failed to achieve competitive results. Deng et al. [7] proposed TIT training strategy, which views SNN with short time steps as pre-trained model for training long time steps SNN. TIT strategy does expedite the convergence of SNN, but it still suffers from the high cost of tuning process and only fits with the TET function introduced in [7]. Meng et al. [22] analyzed the relationship between inputs and outputs of each layer in SNN and proposed DSR method. This method enables fast training of SNNs with long time steps, but it needs two additional measures to achieve good performance. Inspired by [22], we devise SSF method. Our method differs by speeding up the training of SNNs in an easy-to-implement way while producing high accuracy without any other measures.

3. Method

This section first introduces the Leaky-Integrate-and-Fire (LIF) neuron model used in our method. Then, the backward propagation of classic SG is analyzed and revealing the main reason for the rapid growth of computation cost. Based on this discovery, our SSF method is proposed and presented in detail. Finally, to show our method's universality, soft reset function is discussed, along with the application of SSF on it.

3.1. LIF Neuron Model

Different from ANN, neurons in SNN simulate biological neurons' action potential progress, which accumulates inputs as membrane potential and generates a binary spike when exceeding thresholds. As a widely used implement, LIF model can be simply described as [12]:

$$\tau\frac{du}{dt} = -u + RI, u < V_{th}, \tag{1}$$

$$u = u_{reset} \;\&\; fire\ a\ spike, u \geq V_{th}, \tag{2}$$

where u is the accumulated membrane potential, u_{reset} is the resting potential that is usually set as 0, RI stands for inputs, τ is a time constant deciding the decreasing speed of u, V_{th} represents the firing threshold. Eq. (1) figures the membrane potential accumulation process below V_{th}. Eq. (2) means when u is up to V_{th}, the neuron fires a spike and sets the membrane potential to the resting potential u_{reset}, which named as hard reset.

It's worth noting that the differential representation with continuously varying electricitys is not easy to implement in mainstream machine learning frameworks. Wu et al. [36] proposed an iterative model, which can be conducted as:

$$u[t] = (1 - \frac{dt}{\tau})u[t-1] + \frac{dt}{\tau}RI, \tag{3}$$

where $u[t]$ means the membrane potential at time step t . Factor $(1-\frac{dt}{\tau})$ can be simplified as λ, standing for the attenuation in each time step. Input $\frac{dt}{\tau}RI$ equals to the weighted summation of pre-synaptic signals $\sum_j w_j o_j[t]$, where w_j denotes the connection weight between the j-th pre-neuron and current neuron, while $o_j[t]$ means the binary spike from the j-th pre-neuron at time step t. With these settings, the iterative model can be updated as [22]:

$$u[t] = \lambda u[t-1](1 - o[t-1]) + \sum_j w_j o_j[t], \tag{4}$$

$$o_i^l[t] = H(u_i^l[t] - V_{th}), \tag{5}$$

where H denotes the Heaviside function and $u[0] = \sum_j w_j o_j[0]$.

It's worth noting that another widely used IF model equals to LIF model with λ set to 1, so our method can easily adapt to IF models. Besides, SSF is also compatible with soft reset function, we will discuss it in Sec. 3.4.

3.2. Formulation of Backward Process

In this part we analyze the backward process of classic SG method. We use the LIF model introduced above. To simplify our derivation, we set $I_i^l[t] = \sum_j w_{ij} o_j^{l-1}[t]$, representing the input of neuron i in layer l on time step t. The forward process of neuron i in layer l can be described as:

$$u_i^l[t] = \begin{cases} \lambda u_i^l[t-1](1 - o_i^l[t-1]) + I_i^l[t],\ t \neq 0, \\ I_i^l[0],\ t = 0, \end{cases} \tag{6}$$

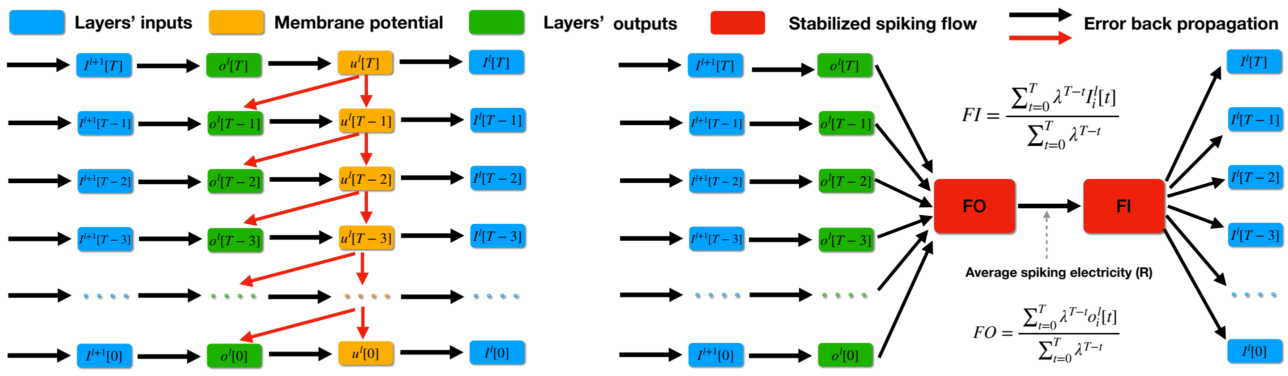
$$o_i^l[t] = H(u_i^l[t] - V_{th}). \tag{7}$$

With the effect of chain rule, we can assume that the gradient of $I_i^{l+1}[t]$ has already obtained through above layers. As the output of last time step, $o_i^l[T]$ ends the whole spiking process, so it's gradient can be generated from $I_i^{l+1}[T]$ directly. For time step $t-1$, $o_i^l[t-1]$ takes part in the accumulation of $u_i^l[t]$. From Eq. (6) and Eq. (7), we have:

$$\begin{aligned} \frac{\partial L}{\partial o_i^l[t-1]} &= \sum_j \frac{\partial L}{\partial I_j^{l+1}[t-1]} \frac{\partial I_j^{l+1}[t-1]}{\partial o_i^l[t-1]} + \\ &\sum_j \frac{\partial L}{\partial I_j^{l+1}[t]} \frac{\partial I_j^{l+1}[t]}{\partial o_i^l[t]} \frac{\partial o_i^l[t]}{\partial u_i^l[t]} * (-\lambda u_i^l[t-1]). \end{aligned} \tag{8}$$

With $\frac{\partial L}{\partial o_i^l[t]}$, we can move on calculating $\frac{\partial L}{\partial u_i^l[t]}$. Same with $o_i^l[T]$, $\frac{\partial L}{\partial u_i^l[T]}$ can be generated from $\frac{\partial L}{\partial o_i^l[T]}$ directly. For other time step $t < T$, $u_i^l[t]$ takes part in the accumulation of $u_i^l[t+1]$. From Eq. (6) and Eq. (7), we have:

$$\frac{\partial L}{\partial u_i^l[t]} = \frac{\partial L}{\partial o_i^l[t]} \frac{\partial o_i^l[t]}{\partial u_i^l[t]} + \frac{\partial L}{\partial u_i^l[t+1]} * \lambda(1 - o_i^l[t]). \tag{9}$$


Layers' inputs
Membrane potential
Layers' outputs
Stabilized spiking flow
Error back propagation
FO
FI
Average spiking electricity (R)
$FI = \frac{\sum_{t=0}^{T}\lambda^{T-t}I_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}}$
$FO = \frac{\sum_{t=0}^{T}\lambda^{T-t}o_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}}$

(a) Backward process of classic SG method.

(b) Backward process of proposed SSF.

Figure 2. Illustration of backward process in classic SG method and the proposed SSF approach. (a) demonstrates classic SG's back-propagation in time domain. The accumulation of membrane potential (i.e., entangled $u^l[t]$) results in exponential growth of training time when the number of time steps increases. As shown in (b), SSF approximates output and input spike trains with stabilized spiking flows and builds the mappings between them with the average spiking electricity, skipping the back propagation in time domain and vastly decreasing the training time cost.

As shown in Eq. (6), $I_i^l[t]$ only connects to $u_i^l[t]$. With $\frac{\partial L}{\partial u_i^l[t]}$, we can have:

$$\frac{\partial L}{\partial I_i^l[t]} = \frac{\partial L}{\partial u_i^l[t]}\frac{\partial u_i^l[t]}{\partial I_i^l[t]} = \frac{\partial L}{\partial u_i^l[t]}, \tag{10}$$

which will be back propagated to the next layer.

Intuitively, Fig. 2a illustrates this backward process, from which we can find that the main computation cost occurs in the back-propagation of membrane accumulation process. What's more, the error back-propagation in time domain is serial, which forces programs to calculate T times in sequence and greatly reduces the efficiency of parallel computing devices like GPU. In fact, as shown in Fig. 1, training time experiences an exponential growth when time steps increase. This raises a natural question: is it possible to avoid the back propagation of membrane potential while not degrade final performance? We answer this question by proposing the SSF method.

3.3. Stabilized Spiking Flow

Due to the complex dynamics of spiking neurons, it's challenge to skip membrane potential and map the conversion of each time step's inputs and outputs while keeping performance consistent. However, from the perspective of global, we find a possible way to approach this goal.

When training SNNs on static datasets like CIFAR-10, a 2D image would be simply repeated T times as input. This kind of input brings a series of repeated spiking and results in a spike train with stable spiking frequencies, which we name as a stabilized spiking flow. She et al. [27] discussed the dynamic of SNN and found that when taking several stabilized spiking flows as inputs, neuron of SNNs would transmit them to another stabilized spiking flow. So when trained on static datasets, SNNs' inputs and outputs of each layer all can be viewed as stabilized spiking flows. Consequently, spiking neurons can be viewed as simple transverters between different stabilized flows with no need considering the complex process of membrane potential accumulation inside them. So, if we can build the translation between these stabilized inputs and outputs, we may skip the accumulation of membrane potential and reduce the exponential training time cost. Inspired by [22], we implement it through the whole input and output electricity.

Formally, considering the decay of membrane potential on each time step, the whole input electricity of a neuron i in layer l can be formulated as $\sum_{t=0}^{T}\lambda^{T-t}I_i^l[t]$. By multiplying both sides of Eq. (6) with the attenuation coefficient and summing over time steps, we get:

$$\begin{aligned}\sum_{t=0}^{T}\lambda^{T-t}u_i^l[t] = &\sum_{t=0}^{T}\lambda^{T-t}I_i^l[t] \\ &+ \sum_{t=0}^{T-1}\lambda^{T-t}u_i[t](1-o_i^l[t]).\end{aligned} \tag{11}$$

Eq. (11) can be further divided to:

$$u_i^l[T](1-o_i^l[T]) = \sum_{t=0}^{T}\lambda^{T-t}(I_i^l[t] - o_i^l[t]u_i^l[t]). \tag{12}$$

The left side of Eq. (12) means electricity left in neuron after time step T, while the right side represents the whole input electricity minus the whole output electricity. Since the remaining electricity $u_i^l[T](1-o_i^l[T])$ is less than V_{th}, with long enough time steps, we can dismiss it and have:

$$\sum_{t=0}^{T}\lambda^{T-t}I_i^l[t] \approx \sum_{t=0}^{T}\lambda^{T-t}o_i^l[t]u_i^l[t]. \tag{13}$$

Eq. (13) describes the relationship between the whole input and output electricity of a neuron. As we mentioned before, stabilized spiking flow equals to stable input or output electricity each time step, so we define stabilized input electricity FI as a constant satisfying $\sum_{t=0}^{T}\lambda^{T-t}FI = \sum_{t=0}^{T}\lambda^{T-t}I_i^l[t]$.

For stabilized output electricity, it connects with membrane potential of those time steps generating spikes. However, $u_i^l[t]$ plays the role of lost electricity when generating a spike. Meanwhile, spiking flow equals to a series of repeated spiking process which includes the same membrane potential in each time step. So electricity lost in each spiking process can be viewed as a stable value R, which can be calculated during inference and used as a constant in backward process. We discussed the effect of R's precision in Sec. 4.5. In our method, we define average spiking electricity as $R = \frac{\sum_{t=0}^{T}\lambda^{T-t}o_i^l[t]u_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}V_{th}o_i^l[t]}$ and define stabilized output flow as $FO = \frac{\sum_{t=0}^{T}\lambda^{T-t}o_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}}$. R would be generated during inference and treated as a constant in backward progress, thoroughly removing the influence of membrane potential.

With these definition and the dynamic of SNN's neurons, the transition between FO and FI can be described as:

$$FO = max(0, \frac{FI}{R * V_{th}}). \tag{14}$$

Considering that too much electricity lost in each spiking process may cause information loss, we set a limit to R by setting $\frac{\partial FO}{\partial FI} = 0$ when $R > 2$.

Through FO and FI, the whole backward progress can be described as:

$$\frac{\partial L}{\partial I_i^l[t]} = \frac{\partial L}{\partial FO}\frac{\partial FO}{\partial FI}\frac{\partial FI}{\partial I_i^l[t]}, \tag{15}$$

where $\frac{\partial L}{\partial FO} = \frac{\sum_{t=0}^{T}\lambda^{T-t}\frac{\partial L}{\partial I_i^{l+1}[t]}}{\sum_{t=0}^{T}\lambda^{T-t}}$ and $\frac{\partial FI}{\partial I_i^l[t]} = 1$ because input $I_i^l[t]$ naturally experiences $T-t$ time steps' damping during inference. As shown in Fig. 2b , we successfully skip the back-propogation of membrane accumulation with the help of FO and FI.

3.4. Soft Reset Function

Aside from hard reset, soft reset function is also widely used in SNN. After generating a spike, soft reset sets membrane potential to $u[t] - V_{th}$ rather than u_{reset}. Formally, LIF model using soft reset function can be formulated as:

$$u[t] = \lambda\Big(u[t-1] - V_{th} * o[t-1]\Big) + \sum_j w_j o_j[t], \tag{16}$$

$$o_i^l[t] = H(u_i^l[t] - V_{th}). \tag{17}$$

Our method is compatible with soft reset method too. With the same process, we can obtain a relationship between input and output electricity:

$$u_i^l[T] - V_{th}o_i^l[T] = \sum_{t=0}^{T}\lambda^{T-t}\Big(I_i^l[t] - V_{th}o_i^l[t]\Big). \tag{18}$$

The left part of Eq. (18) stands for electricity remaining in spiking neurons after time step T, while the right part represents the whole input electricity minus the whole output electricity, similar to Eq. (12).

Different from hard reset, left electricity can be much higher than V_{th} in Eq. (18), so we can not ignore it. Based on the idea of stabilized spiking flow, when left electricity is beyond V_{th}, stabilized input electricity must exceed V_{th} in each time step. In this situation, a spiking neuron will generate a spike every time step, which means that the output spike train reaches it's upper bound and we have no way to distinguish different input spike trains as they are all beyond output trains' representation ability. Specifically, spiking neuron clamps the whole input electricity to $[0, TV_{th}]$.

Similarly, we define $FI = \frac{\sum_{t=0}^{T}\lambda^{T-t}I_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}}$ and $FO = \frac{\sum_{t=0}^{T}\lambda^{T-t}o_i^l[t]}{\sum_{t=0}^{T}\lambda^{T-t}}$. R is not necessary as electricity lost in each spiking process all equals to V_{th}. Therefore, the relationship between input and output spiking flows can be derived as:

$$FO = Clamp(\frac{FI}{V_{th}}, 0, 1). \tag{19}$$

Other parts of backward process resembles hard reset.

4. Experiments

To validate the effectiveness of our method, we first apply SSF to four representative frameworks of SG on both static and neuromorphic datasets. Then we demonstrate the acceleration advantage of SSF when time steps become longer. Finally, we perform ablation study on the influence of average spiking electricity R defined in Sec. 3.3 and the instantiation of parallel computing.

4.1. Datasets

We evaluate our method on three static datasets, CIFAR-10 [18], CIFAR-100 [18], Tiny-ImageNet [5] and one neuromorphic dataset DVS-CIFAR10 [20].

CIFAR-10 and CIFAR-100 datasets are two widely used datasets in image classification filed. They both contain 60000 color images of which 50000 are for training and 10000 are for test. CIFAR-10 has 10 classes with 6000 images each class, while CIFAR-100 has 100 classes with 600 images each class. When training on these two datasets, we utilize a standard data augmentation strategy same as [34].

Tiny-ImageNet dataset is a subset of ImageNet [5]. It contains 200 classes with 500 training samples and 50 validation samples each class. We use a classic transform process, which resizes each picture to 224*224 before horizontal flip and normalization.

Table 1. Classification performance on CIFAR-10,CIFAR-100, Tiny-ImageNet and DVS-CIFAR10. For the first three datasets, we divide the frameworks into 3 classes: ANN, ANN-to-SNN, and SG. Different types of methods are separated by horizontal lines.We carefully rebuild three representative SG frameworks and accelerate their training process with our method. Classification accuracy and backward time spent in one epoch are shown. We use a gray background to distinguish accelerated frameworks from original frameworks and compute speed-up ratios with backward time.

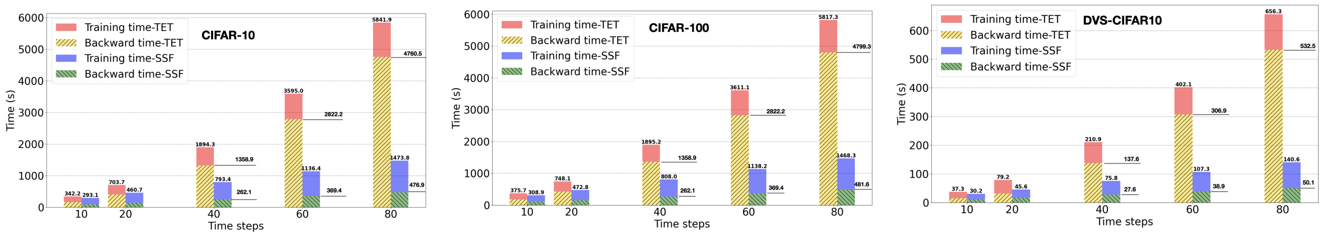
	Method	Network	Time steps	Accuracy	Backward time (s)	Speed-up ratio
CIFAR-10	ANN [22]	PreAct-ResNet-18	/	95.41%	/	/
	ANN-to-SNN [13]	VGG-16	2048	93.63%	/	/
	ANN-to-SNN [39]	VGG-like	600	94.20%	/	/
	ANN-to-SNN [3]	ResNet-18	16	95.92%	/	/
	tdBN [41]	ResNet-19	20	93.07%	651.8	1.00×
	tdBN-ours	ResNet-19	20	93.04%	313.3	**2.08**×
	RecDisN [12]	PreAct-ResNet-18	20	94.60%	406.3	1.00×
	RecDis-ours	PreAct-ResNet-18	20	94.30%	155.6	**2.62**×
	TET [7]	PreAct-ResNet-18	20	95.00%	402.5	1.00×
	TET-ours	PreAct-ResNet-18	20	94.90%	143.0	**2.81**×
CIFAR-100	ANN [22]	PreAct-ResNet-18	/	78.12%	/	/
	ANN-to-SNN [6]	ResNet-20	400-600	69.82%	/	/
	ANN-to-SNN [13]	VGG-16	768	70.09%	/	/
	ANN-to-SNN [39]	VGG-like	300	71.84%	/	/
	tdBN [41]	ResNet-19	20	69.54%	511.0	1.00×
	tdBN-ours	ResNet-19	20	69.19%	254.4	**2.01**×
	RecDis [12]	PreAct-ResNet-18	20	73.97%	403.4	1.00×
	RecDis-ours	PreAct-ResNet-18	20	75.48%	153.0	**2.62**×
	TET [7]	PreAct-ResNet-18	20	74.41%	429.4	1.00×
	TET-ours	PreAct-ResNet-18	20	74.87%	154.8	**2.77**×
TinyImageNet	ANN [33]	ResNet-34	/	57.63%	/	/
	ANN-to-SNN [11]	DCT-SNN	125	52.43%	/	/
	ANN-to-SNN [30]	ResNet-17	5	56.91%	/	/
	SEW [9]	SEW-ResNet34	20	58.32%	7231.4	1.00×
	SEW-ours	SEW-ResNet34	20	58.81%	2250.3	**2.95**×
DVS-CIFAR10	tdBN [41]	ResNet-19	20	67.0%	196.4	1.00×
	tdBN-ours	ResNet-19	20	67.4%	97.6	**2.01**×
	RecDis [12]	VGG-11	20	74.6%	32.8	1.00×
	RecDis-ours	VGG-11	20	74.5%	17.3	**1.90**×
	TET [7]	VGG-11	20	78.7%	31.2	1.00×
	TET-ours	VGG-11	20	74.1%	16.4	**1.90**×
	TET-CE-ours	VGG-11	20	78.0%	15.1	**2.07**×

DVS-CIFAR10 dataset is a neuromorphic dataset based on CIFAR-10. It consists of 10000 event streams in 10 classes which are recorded by event camera .To reduce calculating complexity, we downsample the original 128×128 event streams to 48×48 and select the last 10% of streams of each class for test while the remaining 90% for training.

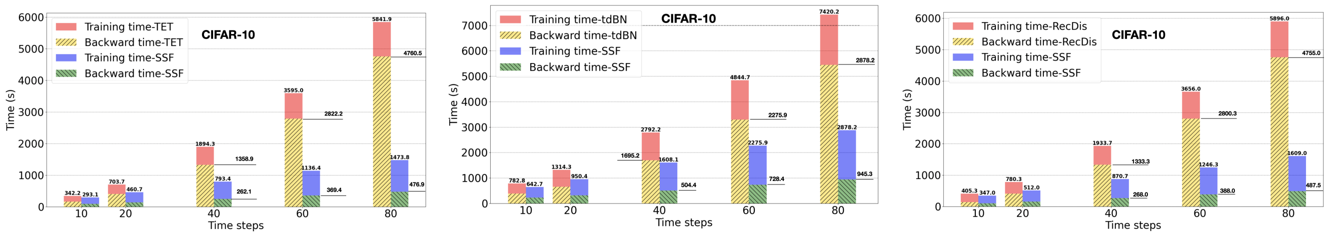
4.2. Implementation Details

We apply SSF to four representative SG frameworks, namely TET [7], RecDis [12], SEW [9] and tdBN [41]. We rebuild these four frameworks as faithfully as possible by following each corresponding paper. Specifically, when SSF is adopted to tdBN, we use the ResNet-19 structure in all cases. As for TET and RecDis, we utilize the PreAct-ResNet-18 structure on CIFAR-10 and CIFAR-100 datasets, while employ the VGG-11 structure on DVS-CIFAR10. SEW framework is only used in Tiny-ImageNet dataset. Unless specified, time steps of all experiments are set to 20. More details are shown in supplementary materials.

In addition to comparing the classification accuracy , we also record two kinds of training time spent in each epoch. One named as training time, consists of time spent in both forward and backward process. The other one named as backward time, only includes time spent in backward process. We run 10 epochs and take the average time spent in each epoch as final results. To ensure the stability of the environment, all time records are obtained through a sever with 8 TITAN X GPUs, while the batch size is kept to 32.



(a) Speed up effects of TET frameworks with varying time steps on different datasets.



(b) Speed up effects of different frameworks with varying time steps on CIFAR-10.

Figure 3. Acceleration effect with varying time steps. In (a), we take TET as an example and illustrate the acceleration effect across different datasets. In (b), all frameworks are evaluated on CIFAR-10 datasets.

Table 2. Experimental results compared to DSR. All experiments are conducted on PreAct-ResNet-18 network with IF model. DSR-noF represents DSR removing firing mechanism modification.

Dataset	Time Step	DSR	DSR-noF	SSF
CIFAR-10	20	95.4%	92.9%	95.4%
	10	94.7%	90.8%	95.3%
	5	94.5%	84.0%	95.0%
CIFAR-100	20	78.5%	74.9%	77.5%
	10	76.8%	62.7%	76.6%
	5	74.4%	53.6%	74.2%

4.3. Comparisons with State-of-the-art

We first quantitatively compare our approach with the four representative training frameworks in terms of classification accuracy and acceleration effect. Results are shown in Table 1, where '*-ours' means accelerating the training of framework '*' with SSF.

When training on static datasets, we can find that all four candidate methods accelerated with SSF consistently achieve the classification accuracy on par with the original counterparts. This indicates that our acceleration strategy is performance-friendly on static datasets. As for neuromorphic dataset, SSF method still obtains competitive performance in RecDis and tdBN frameworks, but appears a bit maladjustment to TET method. This is mainly because the input from DVS-CIFAR10 changes at every time step and is difficult to approximate with stabilized spiking flow, while TET highlights the loss of each time step rather than average output of all time steps. To prove this, we modify the TET loss function to CrossEntry loss in TET framework and obtain an accuracy of 78%, nearly the same with original TET.

While keeping competitive performance, our method significantly decreases training time in all cases. Specifically, on the three static dataset, SSF speeds up the backward process more than twice. In terms of neuromorphic dataset, though these original training methods spend less time in the backward process compared with that on static datasets, SSF can still half the backward time, again demonstrating the universality and effectiveness of our approach.

To analysis the acceleration characteristic from a global view, we also record the accuracy curves of RecDis and RedDis-ours trained on CIFAR-10. Results are shown in supplyment materials, from which we can observe that epochs needed to converge are nearly the same between the original version and the accelerated one.

Our method also works on SNNs with soft reset function, corresponding experiments are shown in supplyment materials. However, we notice that the basic theory of SSF suits hard reset more than soft reset function. As we know, DSR is one of the most success method using relationship between input and output electricity accelerating the training of SNNs with soft reset function. However, the appealing performance of DSR builds on the basis of combining firing mechanism modification and threshold training. We conduct experiments on a network same with DSR, and results are shown in Tab. 2. DSR-noF stands for DSR without firing mechanism modification, which fails achieving good performance. In contrast, SSF doesn't need any additional measures to achieve high performance while speeds up the training process with higher ratio ($2.75\times$ for SSF and $2.69\times$ for DSR with time steps equaling to 20 on CIFAR10).

4.4. Effect of Large Time Steps

As shown in Fig. 4, longer time steps leads to better performance. However, as we derived before, long time steps brings unacceptably high training time. In this section, we study the acceleration effect of our method when time steps grow longer, ranging from 10 to 80.

Fig. 3 illustrates the relationship between training time

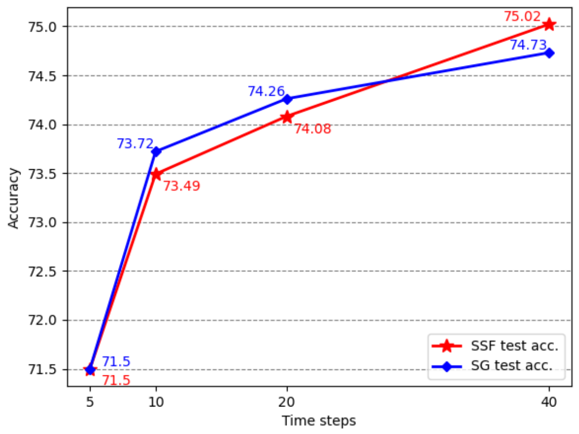


Figure 4. Performance with long time steps. All experiments are conducted in TET framework on CIFAR-100 based on IF model.

and time steps in terms of different methods trained on different datasets. In general, we can find that in classic SG method, backward time grows rapidly and quickly occupies the main part of the training time as the number of time steps increase. By contrast, when accelerating with SSF, both backward time and training time increase in a very low speed, making the acceleration advantage of SSF more meaningful. Specifically, SSF achieves about 10× speed-up ratio when time steps equal to 80.

Fig. 3a shows the growth of training time in the TET framework but on different datasets when time steps grow longer. Across all datasets, the trend that backward time of the original TET grows exponentially with varying time steps remains the same. However, on different datasets with the same exemplar framework, our method can speed up training consistently. These results show that the acceleration effect of SSF does not depend on datasets.

Fig. 3b depicts the growth of training time as increasing number of time steps on the same dataset but with different frameworks. The exponential growing of training time, encountered by all methods in the same setting, indicates the common limitation of SGs on efficient learning. Here again, our SSF endows all the training process with high efficiency, regardless of which framework is employed.

4.5. Ablation Study

We first conduct ablation study on the average spiking electricity R defined in Sec. 3.3. To prove it's importance, we remove R from our method by setting $R = 1$ directly, which eliminates the influence of R in Eq. (14). We also investigate the influence of changeable R with different precision. In our method, we define $R \in \mathbb{R}^{B\times C\times H\times W}$ by default, where B denotes batch size, C represents the number of channels, H and W stands for the height and width of character images in each layer. We adjust the precision of R by reducing it's dimension. All experiments are conducted with the PreAct-ResNet-18 structure on CIFAR-10 with same hyper-parameters. Results are shown in Table 3.

Table 3. Ablation study on the average spiking electricity. All experiments are conducted on PreAct-ResNet-18 architecture on CIFAR-10 dataset with the same hyperparameters.

Setting	Accuracy
$R = 1$	10.00%
$R \in \mathbb{R}$	94.41%
$R \in \mathbb{R}^{B}$	60.19%
$R \in \mathbb{R}^{B\times C}$	61.35%
$R \in \mathbb{R}^{B\times C\times H\times W}$	95.19%

When setting R as 1, the resultant SNN performs nearly a random classifier, proving the importance of average spiking electricity. We can also find that among varying precisions, R defined in our methods achieves best performance. This may manifest that the fine-grained manipulation of the mapping between FO and FI is beneficial to maintain performance.

We also study the influence of different numbers of GPUs on SSF's acceleration ratio. All experiments are performed on a sever with 8 TITAN X GPUs in TET framework on CIFAR-10. We set T to 20 in all experiments. Batch size is set to 16 for each GPU, which means that the whole batch size is $16 * n$ when using n GPUs. As shown in Table 4, GPU number has little effect on the speed up ratio of SSF, proving the stability of SSF's acceleration effect.

Table 4. Effect of different numbers of GPUs.

Num. of GPU	1	2	4	8
Backward time (SG)	2742.5	1432.3	713.2	353.2
Backward time (SSF)	902.1	482.3	239.0	119.2
Speed up ratio	3.04	2.97	2.98	2.96

5. Conclusion

In this paper, we first analyze the backward process of classic SG method and find that the main reason for it's high training time cost is the back-propagation of membrane potential. Based on this discovery, we introduce the SSF method, which skips membrane potential during backward by approximating output and input spike trains with stabilized spiking flows and builds relationship between them directly. Experimental results and analysis show that the proposed method significantly reduces the training time cost of SG while achieving nearly the same performance, and the acceleration is more pronounced in training SNNs with long time steps.

6. Acknowledgment

This work was supported in part by the Major Project for New Generation of AI (No. 2018AAA0100400), the National Natural Science Foundation of China (No. 61836014, No. U21B2042, No. 62072457, No. 62006231, No. U21A20515, No. U2003109), and the InnoHK program.

References

[1] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1-4):17–37, 2002. 2

[2] Tong Bu, Jianhao Ding, Zhaofei Yu, and Tiejun Huang. Optimized potential initialization for low-latency spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 11–20, 2022. 2

[3] 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. 2, 6

[4] Yi Chen, Hong Qu, Malu Zhang, and Yuchen Wang. Deep spiking neural network with neural oscillation and spike-phase information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7073–7080, 2021. 2

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 5

[6] Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2021. 6

[7] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *International Conference on Learning Representations*, 2022. 1, 2, 3, 6

[8] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015. 2

[9] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34:21056–21069, 2021. 6

[10] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2661–2671, 2021. 2

[11] Isha Garg, Sayeed Shafayet Chowdhury, and Kaushik Roy. Dct-snn: Using dct to distribute spatial information over time for low-latency spiking neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4671–4680, 2021. 6

[12] Yufei Guo, Xinyi Tong, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Zhe Ma, and Xuhui Huang. Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 326–335, 2022. 2, 3, 6

[13] 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*, pages 13558–13567, 2020. 2, 6

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1

[15] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, and Timothée Masquelier. Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing*, 205:382–392, 2016. 2

[16] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018. 2

[17] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11270–11277, 2020. 2

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[19] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016. 2

[20] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017. 5

[21] Fangxin Liu, Wenbo Zhao, Yongbiao Chen, Zongwu Wang, and Li Jiang. Spikeconverter: An efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 1692–1701, 2022. 2

[22] Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12444–12453, 2022. 2, 3, 4, 6

[23] Yoshifumi Nishi, Kumiko Nomura, Takao Marukame, and Koichi Mizushima. Stochastic binary synapses having sigmoidal cumulative distribution functions for unsupervised learning with spike timing-dependent plasticity. *Scientific Reports*, 11(1):1–12, 2021. 2

[24] Wachirawit Ponghiran and Kaushik Roy. Spiking neural networks with improved inherent recurrence dynamics for sequential learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8001–8008, 2022. 2

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28, 2015. 1

[26] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. 1

[27] Xueyuan She, Saurabh Dash, and Saibal Mukhopadhyay. Sequence approximation using feedforward spiking neural network for spatiotemporal learning: Theory and optimization methods. In *International Conference on Learning Representations*, 2021. 4

[28] Weihao Tan, Devdhar Patel, and Robert Kozma. Strategy and benchmark for converting deep q-networks to event-driven spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9816–9824, 2021. 2

[29] Guangzhi Tang, Neelesh Kumar, and Konstantinos P Michmizos. Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6090–6097, 2020. 2

[30] Jianxiong Tang, Jianhuang Lai, Xiaohua Xie, Lingxiao Yang, and Wei-Shi Zheng. Snn2ann: A fast and memory-efficient training framework for spiking neural networks. *arXiv preprint arXiv:2206.09449*, 2022. 6

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

[32] John J Wade, Liam J McDaid, Jose A Santos, and Heather M Sayers. Swat: A spiking neural network training algorithm for classification problems. *IEEE Transactions on Neural Networks*, 21(11):1817–1830, 2010. 2

[33] Zenghui Wang and Jun Zhang. Incremental pid controller-based learning rate scheduler for stochastic gradient descent. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 6

[34] Hao Wu, Yueyi Zhang, Wenming Weng, Yongting Zhang, Zhiwei Xiong, Zheng-Jun Zha, Xiaoyan Sun, and Feng Wu. Training spiking neural networks with accumulated spiking flow. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10320–10328, 2021. 2, 3, 5

[35] Jibin Wu, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan. A tandem learning rule for effective training and rapid inference of deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021. 2

[36] 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, pages 1311–1318, 2019. 2, 3

[37] Shuiying Xiang, Zhenxing Ren, Yahui Zhang, Ziwei Song, Xingxing Guo, Genquan Han, and Yue Hao. Training a multi-layer photonic spiking neural network with modified supervised learning algorithm based on photonic stdp. *IEEE Journal of Selected Topics in Quantum Electronics*, 27(2):1–9, 2020. 2

[38] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European Conference on Computer Vision*, pages 418–434, 2018. 1

[39] Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10577–10584, 2021. 2, 6

[40] Zhixuan Zhang and Qi Liu. Spike-event-driven deep spiking neural network with temporal encoding. *IEEE Signal Processing Letters*, 28:484–488, 2021. 2

[41] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11062–11070, 2021. 2, 6

[42] Shibo Zhou, Xiaohua Li, Ying Chen, Sanjeev T Chandrasekaran, and Arindam Sanyal. Temporal-coded deep spiking neural network with easy training and robust performance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11143–11151, 2021. 2