Training Spiking Neural Networks with Accumulated Spiking Flow

Hao Wu, Yueyi Zhang,† Wenming Weng, Yongting Zhang
Zhiwei Xiong, Zheng-Jun Zha, Xiaoyan Sun, Feng Wu
University of Science and Technology of China
National Engineering Laboratory for Brain-inspired Intelligence Technology and Application
{wuhao,wmweng,zytabcd}@mail.ustc.edu.cn, {zhyuey,zwxiong,zhazj,sunxiaoyan,fengwu}@ustc.edu.cn

Abstract

The fast development of neuromorphic hardwares promotes Spiking Neural Networks (SNNs) to a thrilling research avenue. Current SNNs, though much efficient, are less effective compared with leading Artificial Neural Networks (ANNs) especially in supervised learning tasks. Recent efforts further demonstrate the potential of SNNs in supervised learning by introducing approximated backpropagation (BP) methods. To deal with the non-differentiable spike function in SNNs, these BP methods utilize information from the spatio-temporal domain to adjust the model parameters. With the increasing of time window and network size, the computational complexity of spatio-temporal backpropagation augments dramatically. In this paper, we propose a new backpropagation method for SNNs based on the accumulated spiking flow (ASF), i.e. ASF-BP. In the proposed ASF-BP method, updating parameters does not rely on the spike train of spiking neurons but leverage accumulated inputs and outputs of spiking neurons over the time window, which reduces the BP complexity significantly. We further present an adaptive linear estimation model to approach the dynamic characteristics of spiking neurons statistically. Experimental results demonstrate that with our proposed ASF-BP method, light-weight convolutional SNNs achieve superior performances compared with other spike-based BP methods on both non-neuromorphic (MNIST, CIFAR10) and neuromorphic (CIFAR10-DVS) datasets. The code is available at https://github.com/neural-lab/ASF-BP.

Introduction

Artificial Neural Networks (ANNs) (LeCun, Bengio, and Hinton 2015) have achieved a great success in diverse applications such as computer vision and natural language processing. But the success of ANNs is also accompanied by some serious concerns on their huge demand of computational resources and power consumption. In contrast, biological brain can provide excellent cognitive abilities with ultra-low natural power. The outstanding brain efficacy inspires researchers to explore algorithms that mimic the brain's operating mechanism at the neuron level.

Spiking Neural Networks (SNNs), as the third-generation neural networks (Maass 1997), use biologically-realistic but simplified models of neurons to carry out computation. Unlike ANNs that are activated using continuous values, SNNs use sparse binary spikes in time and space dimensions in an event-driven manner. The event-driven mechanism in SNNs greatly reduce the occupation of computing resources and avoids consuming excessive resources to a large extent. Along with the development of SNN-specific hardwares, such as Loihi (Davies et al. 2018) and TrueNorth (Merolla et al. 2014), SNNs have received increasing attention in both academia and industry.

Existing SNNs can be roughly divided into two categories, converted SNNs (Cao, Chen, and Khosla 2015; Sengupta et al. 2019) and directly trained SNNs. A converted SNN model is transferred from certain pre-trained ANN model. Its network architecture is usually consistent with the ANN model and its model parameters are transformed from the pre-trained ANN model by scale adjusting and other conversion operations. Consequently, converted SNN models usually lack biological interpret-ability.

For directly trained SNNs, supervised and unsupervised learning are both attractive research topics. For unsupervised learning, the mainstream learning method is the spike timing dependent plasticity rule (STDP) (Caporale and Dan 2008; Song, Miller, and Abbott 2000; Diehl and Cook 2015). STDP uses synaptic plasticity and spike activity to learn features of input data, which is biologically plausible. On the other hand, the supervised SNNs can achieve much better performance given a large number of labeled training data. However, training this kind of SNNs is quite difficult. Due to the non-differentiable spike functions, the error backpropagation (BP) (Rumelhart, Hinton, and Williams 1986) mechanism commonly adopted in conventional ANNs cannot be directly transferred to SNNs.

There are some successful attempts that introduce BP into SNN models, such as STBP (Wu et al. 2018), SLAYER (Shrestha and Orchard 2018), BP-STDP (Tavanaei and Maida 2019; Luo et al. 2019), which achieve good performances on some simple cognitive tasks. Existing spike-based BP methods utilize the gradient descent method (Ruder 2016) to adjust the model parameters. In this process, there are two obstacles that hinder the development of BP in SNN. First, the issue of non-differential activation function needs to be addressed. Generally, there are two ways to address the issue: (1) replacing the activation function with an estimated

†Corresponding Author

model which is differentiable, (2) directly approximating the gradient. Second, the efficiency of the BP algorithms in SNN needs to be improved. For example, if both temporal and spatial information are counted in the BP process, the BP algorithm has to deal with a huge amount of sparse spike trains, which leads to a time-consuming training.

In this paper, we propose a novel and efficient BP algorithm, called ASF-BP, to train SNN models. Unlike the previous direct training algorithms, our proposed BP algorithm utilizes the accumulated spiking flow (ASF) of neurons to construct an equivalent network. Then the calculation of the gradients in the backward process is based on this equivalent network. In this way, our backward process can be accomplished just in one loop, as easy as ANN's backward process. By establishing the relationship between accumulated outputs and accumulated inputs of neurons, ASF-BP avoids the non-differentiable spiking issue. Meanwhile, ASF-BP also simplifies the calculation process of BP and reduces the occupation of computing and storage resources. Furthermore, we propose an adaptive adjustment mechanism to compute the scale factor of accumulated flows to reflect the real neuron dynamics. To validate the effectiveness of ASF-BP, we use the ASF-BP method to train spiking convolutional networks on the datasets MNIST, CIFAR10 and CIFAR10-DVS for the image classification task. Experimental results demonstrate that SNNs with ASF-BP achieve state-of-the-art performances on the three test datesets.

Our key contributions are summarized as follows:

- We propose a novel BP method with accumulated spiking flows to train SNN models, which could save a lot of computing power. Mathematical derivation is provided for validity.
- We develop an adaptive adjustment algorithm for computing scale factors of accumulated flows to reflect the real neuron dynamics.
- Superior image classification performances are achieved on both non-neuromorphic and neuromorphic datasets.

Background and Related Works

SNN Model

The Integrate-and-Fire (IF) neuron model (Burkitt 2006) is a simplified model for neuron dynamics. The model, which is illustrated in Figure 1(a), can be depicted as

$$I(t) = C_{\mathrm{m}} \frac{dV_{\mathrm{m}}(t)}{dt}, \tag{1}$$

where C_{m} is the membrane capacitance, $V_{\mathrm{m}(t)}$ denotes the membrane potential and I denotes the membrane current. In the IF model, each spike from the pre-synaptic neuron is scaled by a synaptic weight before adding to the membrane potential. When an input current is injected into the neuron, the membrane potential will increase until it reaches its threshold V_{th}, at which point a spike fires. After that, the membrane potential is reset to the resting potential, which is usually zero. The spike activation function can be described as

$$f(u) = \begin{cases} 0, & u < V_{th}, \\ 1, & otherwise. \end{cases} \tag{2}$$

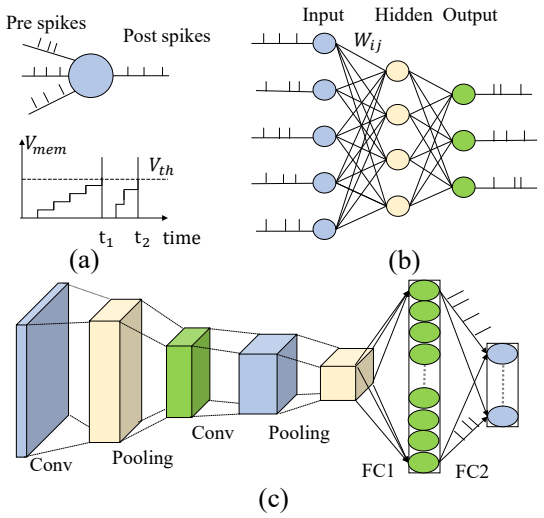

Figure 1: (a) The illustration of the IF model dynamics. The membrane potential of IF neuron increases with the accumulation of weighted spikes. (b) A typical fully connected neural network architecture for SNN. (c) A typical convolution neural network architecture for SNN.

It has been proved that the IF model can approximate ReLU activation function (Tavanaei and Maida 2019; Agatonovic-Kustrin and Beresford 2000), which is very useful in modern SNN architectures. In our work, we utilize IF model as the basic model for network construction and formula derivation. Besides, we extend the application scope of ASF-BP to the LIF model.

As shown in Figure 1(b), a classic multi-layer SNN (Tavanaei et al. 2019) can be separated into three parts: the input layer, the hidden layers and the output layer. The input layer generates spikes for the following layers. For every layer, it mainly output spikes to the next layer. Convolution layers (Krizhevsky, Sutskever, and Hinton 2012) and pooling layers (Boureau et al. 2010) can also be applied in multi-layer SNNs. Figure 1(c) gives an example of SNN which consists of two convolution layers, two pooling layers and two fully connected layers. Mathematically, if a multi-layer SNN only has fully connected layers, the forward process of the SNN can be described as follow

$$\begin{aligned} x_i^{t+1,n} &= \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1}, \\ u_i^{t+1,n} &= u_i^{t,n}(1 - o_i^{t,n}) + x_i^{t+1,n}, \\ o_i^{t+1,n} &= f(u_i^{t+1,n}), \end{aligned} \tag{3}$$

where t is the time index, n and $l(n)$ are the layer index and the number of neurons in the n^{th} layer respectively. w_{ij}^n is the synaptic weight from the j^{th} neuron in the $(n-1)^{th}$ layer to the i^{th} neuron in the n^{th} layer. $x_i^{t,n}$, $u_i^{t,n}$ are the input current and membrane potential of the i^{th} neuron in the n^{th} layer at time t respectively. $o_i^{t,n} \in \{0, 1\}$ is the output spike

generated by the activation function $f(\cdot)$. It should be noted that in our model, biases are not introduced for calculation.

BP Methods

Most SNN related works attempt to solve two problems: credit assignment and non-differentiability of spike function. For the credit assignment problem in SNN, a spatio-temporal training framework based on backpropagation through time (BPTT) is proposed (Wu et al. 2018), which is employed in many SNN works. By considering layer-by-layer spatial domain and timing-dependent temporal domain in the training phase, as well as approximated derivative for spike function, gradient descent training is applied to SNN successfully. Then an explicitly iterative LIF model and a neuron normalization technique is developed in order to adjust neural selectivity (Wu et al. 2019). In addition, Wu et al. give a simple encoding method for SNN to narrow the time window. A probability density function of spike state change is applied to solve the non-differentiability of spike function properly (Shrestha and Orchard 2018). With this approximate derivative, they introduce a BP mechanism for learning both synaptic weights and axonal delays. By building a mathematical model based on the IF model, Lee et al. propose a novel algorithm to solve the non-differentiable problem of LIF model (Lee et al. 2020) and achieve good results on image classification tasks.

The loss function of these methods is usually defined as mean squared error between the average output spikes of output layer and the label of sample $\mathbf{Y}$

$$L = \frac{1}{2}\|\mathbf{Y} - \frac{1}{T}\sum_{t=1}^{T}\mathbf{o}^{t,N}\|_2^2 \tag{4}$$

where $\mathbf{o}^{t,N}$ is the output vector of the output layer N at time step t. With this loss function, the gradient for the weight matrix is given by

$$\frac{\partial L}{\partial \mathbf{W}^n} = \sum_{t=1}^{T}\frac{\partial L}{\partial \mathbf{u}^{\mathbf{t,n}}}\mathbf{o}^{t,n-1T} \tag{5}$$

It can be observed that STBP has to accomplish the backward process in two loops: temporal and spatial. Thus the computational complexity is large and augments with the increasing of time window and network depth.

We recently notice that a tandem framework is proposed for training SNN models (Wu et al. 2020), which is similar to our work. Wu et al. construct a tandem learning framework, which consists of an SNN and an ANN coupled through weight sharing. The ANN in their framework facilitates the error backpropagation for the training of SNN models. Although our work shares some insights with the tandem framework, there are still some differences. The membrane potential of neurons in the tandem framework follows a "reduce by subtraction" way, which is not bio-plausible and different from our setting. In contrast to tandem framework's static activation function, our work utilizes an adaptive adjustment mechanism to rapidly change the neuron dynamics, which is beneficial for the weights updating. Moreover, our input coding scheme is distinct from the tandem framework. All the differences are explained in the following sections.

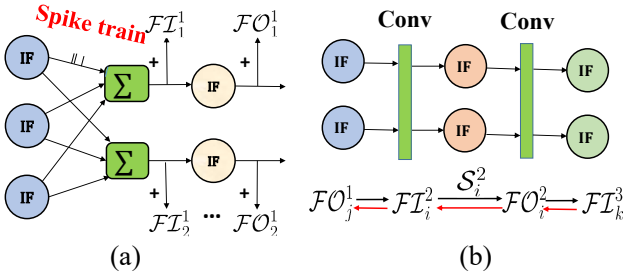

Figure 2: (a) The illustration of the process to calculate the accumulated spiking flow. (b) The illustration of constructed equivalent network. The equivalent network is built based on weights of original spiking neural network and uses accumulated spiking flow as its input and output. The backpropagation of original SNN is substituted by the backpropagation of the equivalent network.

The ASF-BP Method

The Forward Process

The basic model we utilize in our network architecture is similar to other convolutional spiking neural networks (Lee et al. 2020; Shrestha and Orchard 2018; Wu et al. 2019). For convolution layers, the weights of the kernels are based on continuous value and biases are ignored in our settings. For pooling layers, both the max pooling and average pooling are compatible in our network.

Follows the setting in (Lee et al. 2020), the output layer of our network doesn't fire any spikes, which is different from the settings in (Wu et al. 2018). In other words, the output neurons have an infinite threshold and always accumulate input currents. The loss function is defined as

$$L = \frac{1}{2}\|\mathbf{Y} - \frac{\mathbf{u}^{T,N}}{TV_{th}}\|_2^2 \tag{6}$$

where V_{th} is a virtual threshold, T is the length of the time window, $\mathbf{Y}$ denotes the label vector of the sample, $\mathbf{u}^{T,N}$ denotes the membrane potential vector of the output layer.

The Backward Process

Construction of the Equivalent Network For convenience, we assume that neurons in adjacent levels are fully connected. Actually, if the neurons are convolutionally connected, the basic derivation is very similar and will not alter the conclusion. For the backpropagation method, the main target is to find the gradient fast and accurately. Here we introduce the concept of accumulated spiking flow, which is the accumulated inputs and outputs for spiking neurons over the whole time window. Technically, the accumulated spiking flow values can be expressed using the following two equations.

$$\mathcal{FI}_i^n = \sum_{t=1}^{T} x_i^{t,n}, \qquad \mathcal{FO}_i^n = \sum_{t=1}^{T} o_i^{t,n}. \tag{7}$$

Combining Equation (3) and (7), $\mathcal{FI}_i^n$ can be rewritten as

$$\begin{aligned}\mathcal{FI}_i^n &= \sum_{t=1}^{T} x_i^{t,n} = \sum_{t=1}^{T}\sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t,n-1} \\ &= \sum_{j=1}^{l(n-1)} w_{ij}^n \sum_{t=1}^{T} o_j^{t,n-1} = \sum_{j=1}^{l(n-1)} w_{ij}^n \mathcal{FO}_j^{n-1}.\end{aligned} \tag{8}$$

Meanwhile, according to the activation principle, $\mathcal{FI}_i^n$ and $\mathcal{FO}_i^n$ should be approximately proportional to the ratio V_{th}. Figure 2(a) shows the process to calculate the accumulated spiking flow.

To better describe the relationship between the accumulated input and output flow, we define a scale factor $\mathcal{S}_i^n$ for every neuron, which is the ratio of $\mathcal{FO}_i^n$ to $\mathcal{FI}_i^n$. The above-mentioned relationship can be depicted as a linear model $\mathcal{S}_i^n = \frac{\mathcal{FO}_i^n}{\mathcal{FI}_i^n}$.

It should be noted that all the $\mathcal{FI}_i^n$, $\mathcal{FO}_i^n$ and $\mathcal{S}_i^n$ can be calculated in the forward process. In addition, the output of the final layer $mem_i^{T,N}$ is equal to $\mathcal{FI}_i^N$. So after forward process, it is possible to utilize $\mathcal{FI}_i^n$, $\mathcal{FO}_i^n$ and $\mathcal{S}_i^n$ to construct an equivalent network and then develop our back-propagation algorithm.

Derivation of Gradients We use $\mathcal{FO}_i^n$ as the intermediate variable to derive the gradients for the weights. Since $\mathcal{FI}_i^N$ is the final output, there is no $\mathcal{FO}_i^N$ existing. So we calculate the gradients from the $(N-1)^{th}$ layer

$$\begin{aligned}\frac{\partial L}{\partial \mathcal{FO}_i^{N-1}} &= \sum_{j}^{l(N)} \frac{\partial L}{\partial mem^{N,j}} \frac{\partial mem^{N,j}}{\partial \mathcal{FO}_i^{N-1}} \\ &= \sum_j (\frac{mem^{N,j}}{TV_{th}} - Y_j)\frac{w_{ji}^N}{TV_{th}}.\end{aligned} \tag{9}$$

For $n < N-1$, once again using the chain rule, we have

$$\begin{aligned}\frac{\partial L}{\partial \mathcal{FO}_i^n} &= \sum_{j}^{l(n+1)} \frac{\partial L}{\partial \mathcal{FO}_j^{n+1}} \frac{\partial \mathcal{FO}_j^{n+1}}{\partial \mathcal{FI}_j^{n+1}} \frac{\partial \mathcal{FI}_j^{n+1}}{\partial \mathcal{FO}_i^n} \\ &= \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^{n+1}} \mathcal{S}_j^{n+1} w_{ji}^{n+1}.\end{aligned} \tag{10}$$

Furthermore, the weight updating rule is expressed as below

$$\begin{aligned}\frac{\partial L}{\partial w_{ji}^n} &= \frac{\partial L}{\partial \mathcal{FO}_j^n}\frac{\partial \mathcal{FO}_j^n}{\partial \mathcal{FI}_j^n}\frac{\partial \mathcal{FI}_j^n}{\partial w_{ji}^n} = \frac{\partial L}{\partial \mathcal{FO}_j^n}\mathcal{S}_j^n \mathcal{FO}_i^{n-1}, \\ w_{ji}^n &= w_{ji}^n - \eta \frac{\partial L}{\partial w_{ji}^n},\end{aligned} \tag{11}$$

where η is the learning rate. Figure 2(b) illustrates the constructed equivalent network as well as the forward and backward processes.

Adaptive Mechanism for Scale Factors In practice, the scale factor $\mathcal{S}_i^n$ is not stable. For a neuron with sparse spike input, the value of $\mathcal{S}_i^n$ might vary dramatically. Actually, the deployment of scale factors for the neurons assumes there is a linear model for the spiking activation. We deem neurons in the same layer should have similar distributions of input strength which determine the value of scale factor. Given this observation, we apply a uniform scale factor $\mathcal{S}_u^n$ for all the neurons in the same layer. Thus Equation (9) is updated and expressed as

$$S_u^n = \frac{\sum_i^{l(n)} \mathcal{FO}_i^n}{\sum_i^{l(n)} \mathcal{FI}_i^n}. \tag{12}$$

To keep the stabilization, we update the scale factors every a few epochs. In-depth analysis of adaptive mechanism's benefits is given in the experiment section. The utilization of adaptive mechanism is a key difference from (Wu et al. 2020), while Wu et al. only exploit a static activation function to construct the ANN.

Finally, we show the details of the ASF-BP with the pseudo-code in Algorithm 1. It should be noted that although our proposed method is initially based on the IF model, we can still apply the algorithm to the LIF model with minor changes.

Algorithm 1 Training process of ASF-BP

Require: Network input $input_i^t$, sample label $\mathbf{Y}$, parameters and states of network $w_{ji}^n, u_i^{t,n}, o_i^{t,n}$, t = 1,2,...T, n = 1,2,...N

Ensure: Update of network parameters

Initialization

$u_i^n, \mathcal{FI}_i^n, \mathcal{FO}_i^n \leftarrow 0$, $w_{ji}^n \leftarrow N(\mu, \sigma^2)$

Forward

for t = 1 to T **do**

  $o_i^{t,1} \leftarrow$Poisson$(input_i^t)$

  for n = 1 to N **do**

    $x_i^{t,n} \leftarrow \mathbf{w} \cdot \mathbf{o}^{t,n-1}$

    $(u_i^{t,n}, o_i^{t,n}) \leftarrow$ Update$(x_i^{t,n}, u_i^{t-1,n})$ ▷ Eq(3)

    $\mathcal{FI}_i^n \leftarrow \mathcal{FI}_i^n + x_i^{t,n}$,

    $\mathcal{FO}_i^n \leftarrow \mathcal{FO}_i^n + o_i^{t,n}$

  end for

end for

CalculateScale

for n = 1 to N **do**

  $\mathcal{S}_u^n = \sum_i \mathcal{FO}_i^n / \sum_i \mathcal{FI}_i^n$ ▷ Eq(13)

end for

Loss

L $\leftarrow$ CalculateLoss$(Y, \mathcal{FI}^N)$ ▷ Eq(6)

Backward

Gradient initialization: $\frac{\partial L}{\partial w_{ji}^n} \leftarrow 0$

Calculate $\frac{\partial L}{\partial \mathcal{FO}_j^{N-1}}$ and $\frac{\partial L}{\partial w_{ji}^{N-1}}$ ▷ Eq(10)(12)

for n = N-2,..,1 **do**

  $\frac{\partial L}{\partial \mathcal{FO}_i^n} \leftarrow \sum_j \frac{\partial L}{\partial \mathcal{FO}_j^{n+1}} \mathcal{S}_u^{n+1} w_{ji}^{n+1}$,

  $\frac{\partial L}{\partial w_{ji}^n} \leftarrow \frac{\partial L}{\partial \mathcal{FO}_j^n} \mathcal{S}_u^n \mathcal{FO}_i^{n-1}$ ▷ Eq(11)(12)

end for

Update Weights

Experiments

Datasets

Two kinds of image datasets are utilized to test the ASF-BP method for image classification tasks. They are non-neuromorphic datasets MNIST (LeCun et al. 1998), CIFAR10 (Alex Krizhevsky 2009) and neuromorphic dataset CIFAR10-DVS (Li et al. 2017).

MNIST is a handwritten digital dataset, which is a standard benchmark used to evaluate the performance of pattern recognition and machine learning algorithms. The dataset contains 70,000 grayscale images in 10 classes, of which 60,000 images are used for training and 10,000 images are for testing. The resolution of the images is 28×28. Poisson sampling is used to generate spikes. There is no data augmentation utilized for MNIST dataset. In the output layer, the neuron with the highest membrane potential is treated as the target neuron.

CIFAR10 dataset is widely used in machine learning research for object classification. It contains 60,000 color images in 10 classes, of which 50,000 images are for training and 10,000 images are for testing. The resolution of CIFAR10 images is 32×32. For CIFAR10 images, we utilize a standard data augmentation strategy, which consists of random crop, random horizontal flip and normalization. Poisson sampling is also applied to generate spike trains.

Here it should be emphasized that (Wu et al. 2019, 2020) don't use spikes as the network input. They directly feed the network with the real values from the image, which is not bio-plausible. That might explain why they can achieve good results in a short time window.

CIFAR10-DVS is a neuromorphic dataset based on CIFAR10 dataset, which consists of 10000 event streams in 10 classes. The event stream is recorded by the event-based camera when the images are moving slowly. The event-based camera used in the dataset is with the resolution 128×128. There are two kinds of events for the spikes: ON events and OFF events. To reduce calculating complexity, we downsample the event stream to the resolution 42×42. Thus for every time step, the dimension of the input data is $42 \times 42 \times 2$. We randomly select 90% of the images as the training images and the rest images are the testing images for every class. Actually, event-based cameras are very sensitive to object motions and generate the spikes in micro-second level temporal resolution. In practice, we use a time span of 12 ms to accumulate the original spikes.

Implementation Details

We implement two convolutional spiking neural networks LeNet (LeCun et al. 1998) and VGG7 (Simonyan and Zisserman 2014) for these datasets. LeNet has two convolution layers, two pooling layers and two fully connected layers, while VGG7 has five convolution layers, two pooling layers and two fully connected layers. For both networks, we employ two neural models: IF model and LIF model. For the LIF model, we utilize 0.99 as the decay coefficient. The simulation code is written with the Pytorch framework (Paszke et al. 2017) , which provides easy interfaces for GPU acceleration and auto differentiation. All the models are trained using one NVIDIA TitanXP Graphics card.

Model	Accuracy
(Lee et al. 2016)	99.31%
(Wu et al. 2018)	99.42%
(Jin, Zhang, and Li 2018)	99.49%
(Zhang and Li 2019)	99.62%
(Lee et al. 2020)	99.59%
This work (IF)	**99.65%**
This work (LIF)	**99.60%**

Table 1: Comparison of different models on MNIST.

The weights of the convolution layer and the fully connected layer of our networks are initialized according to the normal distribution (He et al. 2015). The threshold of neurons is tuned according to different types of networks and datasets, which is typically set between 0.5 and 2. We adopt the Adam optimizer (Kingma and Ba 2014) to adjust the learning rate with initial lr = 8.5×10^{-4} or 5×10^{-4} for different datasets. After dozens of epochs, we will decrease the learning rate manually. The batch size is set to 60, 100, 40 for the MNIST, CIFAR10 and CIFAR10-DVS datasets respectively. The scale factor used in backward process is updated every 5 epochs.

Performance

MNIST Dataset The LeNet network is used to evaluate the performance of our ASF-BP algorithm on the MNIST dataset. Table 1 compares the performance of ASF-BP with other leading methods on the MNIST dataset. It can be observed that our proposed method achieves the state-of-the-art performance. It is necessary to point out that the time windows of (Zhang and Li 2019) and (Jin, Zhang, and Li 2018) in the experiment are 400. In addition, we compare the training time of our ASF-BP method and the spatio-temporal based BP method (Lee et al. 2020) on the MNIST dataset in the following section.

CIFAR10 Dataset For CIFAR10 dataset, the VGG7 network is applied to evaluate the performance. Figure 4(b) shows the accuracy variation using different lengths of time window for our ASF-BP method on the CIFAR10 dataset. It can be seen that the accuracy of our framework grows as the length of time window increases. Table 2 compares the performance of other leading algorithms on CIFAR10. It can be seen that ASF-BP achieves 91.35% accuracy which outperforms the performance of other methods when time window is set to 400. We also compare the training time of our ASF-BP with the method used in (Lee et al. 2020) on the CIFAR10 dataset in the following section.

It is necessary to point out that the time windows of (Panda and Roy 2016) and (Lee et al. 2020) in the experiment are 250 and 100 respectively. The network used in (Rueckauer et al. 2017) is a converted SNN. To be fair, we also use the ASF-BP with the time window 100 and the best accuracy we can achieve is 89.83%. Though the accuracy is slightly lower

Model	Accuracy
(Panda and Roy 2016)	70.16%
(Wu et al. 2019)	90.53%
(Rueckauer et al. 2017)	90.80%
(Lee et al. 2020)	90.95%
(Wu et al. 2020)	90.98%
This work (IF)	**91.35%**
This work (LIF)	**90.11%**

Table 2: Comparison of different models on CIFAR10

Model	Method	Accuracy
(Orchard et al. 2015)	Random Forest	31.0%
(Lagorce et al. 2017)	HOTS	27.1%
(Sironi et al. 2018)	HAT	52.4%
(Sironi et al. 2018)	Gabor-SNN	24.5%
(Wu et al. 2019)	Spiking CNN (STBP)	60.5%
(Wu et al. 2020)	Tandem Learning	58.65%
This work (IF)	Spiking CNN (ASF-BP)	**62.5%**
This work (LIF)	Spiking CNN (ASF-BP)	**58.2%**

Table 3: Comparison of different models on CIFAR10-DVS

than the result listed in Table 2, it should be noted that the VGG7 network is much simpler than ResNet11 (Lee et al. 2020) and CifarNet (Wu et al. 2019). In addition, (Wu et al. 2019, 2020) use continuous-valued data as input instead of spikes, which is different from our setting.

CIFAR10-DVS Dataset We use the VGG7 network for classification on CIFAR10-DVS, whose structure is similar to what we used in the experiment on the CIFAR10 dataset. The only difference is that the channel numbers of the convolution layers are added. Because the event streams are sparse, to make full use of the data, we repeatedly inject the input data for three times. Table 3 compares the performance of our ASF-BP algorithm and other existing algorithms on CIFAR10-DVS. It can be seen that ASF-BP achieves the comparable performance with other works on the CIFAR10-DVS dataset. It is necessary to mention that (Wu et al. 2019) reduces the temporal resolution by similarly accumulating the spike train within every 5 ms.

Experiment Analysis

Training Time Our proposed ASF-BP method is able to accomplish the BP process in just one loop. Compared with other spatio-temporal methods (Lee et al. 2020; Wu et al. 2018), the calculating complexity is greatly reduced. To explicitly show the acceleration effects of ASF-BP in training SNN, we compare the training time of ASF-BP and the method proposed by Lee et al. in Figure 3. The figure shows that ASF-BP has a 3.0x acceleration on MNIST and CIFAR10 when time window is not very large. Besides, ASF-BP has a larger acceleration ratio (up to 6) as time window goes larger.

To be fair, both methods utilize the same time window, batch size, neuron model and the network architecture during

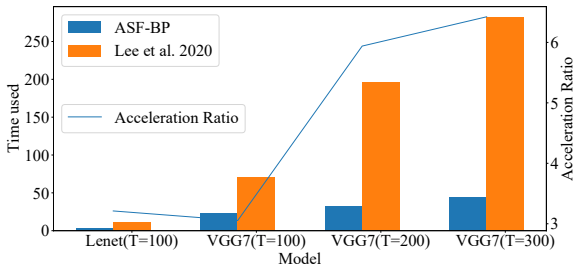

Figure 3: The illustration of the training time comparison between ASF-BP and (Lee et al. 2020) on four experiments. The first experiment is based on MNIST dataset and the remaining experiments are based on CIFAR10 dataset. The bar figure illustrates the training time (in minutes) consumed in an epoch. The acceleration line shows the acceleration ratio to corresponding model.

the comparison. It should be noted that, when training the model with Lee et al.'s method, if the batch size is 50, the GPU memory of single GPU card cannot afford the VGG7 model with 200 and 300 time window. Thus four graphics cards are used to facilitate the training. Since there are overheads for the communication among different GPU cards, the training time increases a lot. From Figure 3, it is easy to observe that ASF-BP is able to accelerate training effectively.

Effect of Scale Factor The neural dynamics of the IF model can be roughly summarized as accumulating input current and firing spikes if the membrane potential surpasses the threshold. In contrast, LIF model has extra leaky mechanism. For both IF and LIF models, the output spike count should be approximately proportional to the total input current during the whole time window. Many SNN training schemes assume that the ratio of spike count to the total input current for a neuron is $1/V_{th}$. However, the membrane at firing time is always larger than the firing threshold. The ratio should be definitely less than $1/V_{th}$. In our paper, we call this ratio scale factor. It is difficult to give a precise value or distribution for the scale factor because it changes a lot with the variance of the input current.

We design an experiment to prove that the scale factor changes as the input current varies. In the experiment, there is one neuron following the IF model. We set the time window to 100. For convenience, we set positive values for weights so that we can get a correct scale factor. During the whole time window, we keep the weight of synapse unchanged and the intensity of input is directly proportional to the continuous-valued pixel. The firing threshold is set to 1. Based on the spiking flow in the experiment, we compute scale factors for input current with different distributions respectively.

Table 4 lists the scale factors under input currents with different distributions. It should be noted that we control the strength of the input by adjusting the intensity of the pixels and keeping weights unchanged here. From the table we can see that the scale factor decreases with the increasing of input current. Therefore, it is easy to conclude that we

Distribution	U(0,0.5)	U(0.1,0.6)	U(0.2,0.7)
Scale Factor	0.641	0.478	0.385
Distribution	U(0.3,0.8)	U(0.4,0.9)	U(0.5,1)
Scale Factor	0.335	0.286	0.249

Table 4: Scale factor variation as input current with different distributions. U(a,b) denotes the uniform distribution between a and b.

need to update the scale factor from time to time to make our equivalent network reflect the real neural dynamics of the original network. It should be mentioned that the updating strategy of scale factor in ASF-BP is different from settings in this experiment. In our ASF-BP, we update the scale factors layer-wisely, not neuron by neuron. We utilize this strategy because there are some extreme values for scale factor, which have negative impacts on network performances. Layer-wise updating in ASF-BP, though not accurate for every neuron, guarantees that the scale factor of most neurons will not deviate the true ratio too much.

Effect of Time Window A single spike could not encode the full input information. In SNN, we usually utilize rate-coded schemes to encode the source information, which converts continuous-valued data to spike trains in a time window. Thus, the length of the time window is directly related to the performance of SNNs. To investigate the relationship between the length of time window and network performance, we design a series of experiments with different time windows on MNIST and CIFAR10 datasets. Figure 4 (a, b) shows the image classification performance variation of our LeNet and VGG7 models with time window from 100 to 500, as the training epochs increase. Figure 4(c) illustrates the image classification performance variation as the time window increases on both MNIST and CIFAR10 datasets.

From all the sub-figures in Figure 4, we can observe that generally the performance of our model gets higher as the time window increases. However, when time window is over 300, the performance will not increase but even decrease to an extent. The possible reason is that the input image pixel value needs a big enough time window to represent the source information. But more time steps are redundant and may mislead the training direction.

Conclusion and Discussion

We have proposed a novel backpropagation method ASF-BP for training SNNs. The fundamental network of ASF-BP is convolutional SNNs based on the IF neuron model. We also extend to LIF model by adding leaky factor to the IF model. When training SNNs, ASF-BP first constructs an equivalent network based on the accumulated spiking flow. Without changing the forward process, the BP algorithm just uses the equivalent network and accomplishes weight updating in single loop. The typical non-differentiable issue of SNNs is avoided due to the employment of accumulated spiking flow and the scale factor. In addition, ASF-BP also takes an adaptive mechanism for the calculation of the scale factor to make the linear estimation closer to the dynamic characteristics of

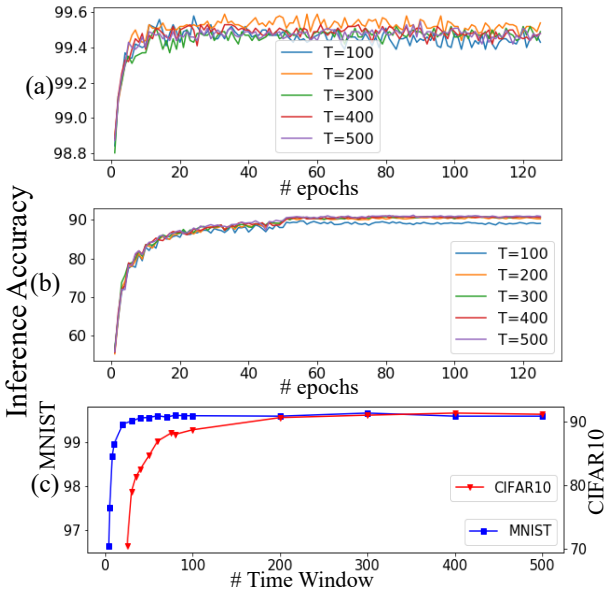

Figure 4: (a) The performance of LeNet network on the MNIST dataset. (b) The performance of VGG7 network on the CIFAR10 dataset. (c) The performance improvement as the time window increases on MNIST and CIFAR10.

spiking neurons statistically. ASF-BP proves its effectiveness by achieving superior performance on both neuromorphic and non-neuromorphic datasets.

ASF-BP is an effective way to train the SNN model, but it still has some limitations. Firstly, ASF-BP updates the weights over the entire time window by capturing spiking flows and ignores the temporal information. For non-neuromorphic datasets, the spike train encodes the intensity values in a rate-coded manner, thus the temporal information is not significant. But for neuromorphic datasets, the spike train embeds more information, such as spike order and spike count, which partially explains why the ASF-BP doesn't achieve satisfactory results (although relatively better) on the neuromorphic dataset. Secondly, ASF-BP could provide a decent mathematically derivation for IF neural models. However, for the LIF neural model, which is more bio-plausible, some of the mathematical expression cannot be well established. In our experiments, we can see that the results for the LIF model is not much worse than the IF model for LeNet and VGG7. But what if the SNN is deeper? In the future, we will continue to investigate how to close the gap between IF and LIF models using ASF-BP.

Acknowledgements

This work was supported in part by National Key R&D Program of China under grant No. 2020AAA0108602 and No. 2020AAA0105700, National Natural Science Foundation of China under grant No. 61901435 and No. 62032006 and Anhui Provincial Natural Science Foundation under grant No. 1908085QF256.

References

Agatonovic-Kustrin, S.; and Beresford, R. 2000. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis* 22(5): 717–727.

Alex Krizhevsky, G. H. 2009. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep* 1.

Boureau, Y.-L.; Bach, F.; LeCun, Y.; and Ponce, J. 2010. Learning mid-level features for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2559–2566.

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

Cao, Y.; Chen, Y.; and Khosla, D. 2015. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision* 113(1): 54–66.

Caporale, N.; and Dan, Y. 2008. Spike timing–dependent plasticity: a Hebbian learning rule. *Annu. Rev. Neurosci.* 31: 25–46.

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

Diehl, P. U.; and Cook, M. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience* 9: 99.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

Jin, Y.; Zhang, W.; and Li, P. 2018. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*, 7005–7015.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*, 1097–1105. Curran Associates, Inc.

Lagorce, X.; Orchard, G.; Galluppi, F.; Shi, B. E.; and Benosman, R. B. 2017. HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(7): 1346–1359.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553): 436–444.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

Lee, C.; Sarwar, S. S.; Panda, P.; Srinivasan, G.; and Roy, K. 2020. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience* 14.

Li, H.; Liu, H.; Ji, X.; Li, G.; and Shi, L. 2017. CIFAR10-DVS: An Event-Stream Dataset for Object Classification. *Frontiers in Neuroscience* 11: 309.

Luo, X.; Qu, H.; Zhang, Y.; and Chen, Y. 2019. First Error-Based Supervised Learning Algorithm for Spiking Neural Networks. *Frontiers in Neuroscience* 13.

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

Merolla, P. A.; Arthur, J. V.; Alvarez-Icaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345(6197): 668–673.

Orchard, G.; Meyer, C.; Etienne-Cummings, R.; Posch, C.; Thakor, N.; and Benosman, R. 2015. HFirst: A Temporal Approach to Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37(10): 2028–2040.

Panda, P.; and Roy, K. 2016. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)*, 299–306. IEEE.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS Workshop*.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* .

Rueckauer, B.; Lungu, I.-A.; Hu, Y.; Pfeiffer, M.; and Liu, S.-C. 2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience* 11: 682.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature* 323(6088): 533–536.

Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; and Roy, K. 2019. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience* 13.

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

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

Sironi, A.; Brambilla, M.; Bourdis, N.; Lagorce, X.; and Benosman, R. 2018. HATS: Histograms of Averaged Time Surfaces for Robust Event-Based Object Classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Song, S.; Miller, K. D.; and Abbott, L. F. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience* 3(9): 919–926.

Tavanaei, A.; Ghodrati, M.; Kheradpisheh, S. R.; Masquelier, T.; and Maida, A. 2019. Deep learning in spiking neural networks. *Neural Networks* 111: 47–63.

Tavanaei, A.; and Maida, A. 2019. BP-STDP: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing* 330: 39–47.

Wu, J.; Chua, Y.; Zhang, M.; Li, G.; Li, H.; and Tan, K. C. 2020. A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks. *arXiv preprint arXiv:1907.01167* .

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

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

Zhang, W.; and Li, P. 2019. Spike-Train Level Backpropagation for Training Deep Recurrent Spiking Neural Networks. In *Advances in Neural Information Processing Systems*, 7800–7811.