Training Spiking Neural Networks with Event-driven Backpropagation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Spiking Neural networks (SNNs) represent and transmit information by spatiotemporal spike patterns, which bring two major advantages: biological plausibility and suitability for ultralow-power neuromorphic implementation. Despite this, the binary firing characteristic makes training SNNs more challenging. To learn the parameters of deep SNNs in an event-driven fashion as in inference of SNNs, backpropagation with respect to spike timing is proposed. Although this event-driven learning has the advantages of lower computational cost and memory occupation, the accuracy is far below the recurrent neural network-like learning approaches. In this paper, we first analyze the commonly used temporal backpropagation training approach and prove that the sum of gradients remains unchanged between fully-connected and convolutional layers. Secondly, we show that the max pooling layer meets the above invariance rule, while the average pooling layer does not, which will suffer the gradient vanishing problem but can be revised to meet the requirement. Thirdly, we point out the reverse gradient problem for time-based gradients and propose a backward kernel that can solve this problem and keep the property of the invariable sum of gradients. The experimental results show that the proposed approach achieves the state-of-the-art performance on CIFAR10 among time-based training methods. Also, this is the first time that the time-based backpropagation approach successfully trains SNN on the CIFAR100 dataset.

1 Introduction

Motivated by the principles of brain computing, Spiking Neural Networks (SNNs) are considered as the third generation of neural networks [1, 2]. SNNs are developed to work in power-critical scenarios, such as edge computing. When run on dedicated neuromorphic chips, they can accomplish the tasks [3, 4, 5] with ultra-low power consumption [6, 7, 8, 9, 10, 11]. In contrast, the last generation of neural networks – Artificial Neural Networks (ANNs) [12], generally require a large amount of computation resource (e.g., GPUs). This advantage of SNNs on power consumption largely relies on efficient event-based computations [13, 14]. Another advantage of SNNs originates from their biological reality (compared to ANNs). The similarity between SNNs and biological brains provides an excellent opportunity to study how the brain computes at the neuronal circuit level [15].

Compared with artificial neural networks, developing supervised learning algorithms for spiking neural networks requires more effort. The main challenge for training SNNs comes from the binary nature of spikes and the non-differentiability of the membrane potential at spike time. This difficulty

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

in training impedes the performance of SNNs in pattern classification tasks compared to their ANN counterparts. Existing supervised learning methods of SNNs can be grouped into two categories:

The first category consists of recurrent neural network (RNN)-like learning algorithms. These algorithms treat spiking neural networks as binary-output recurrent neural networks and handle the discontinuities of membrane potential at spike times with continuous surrogate derivatives [16]. They typically train deep SNNs with surrogate gradients based on the idea of backpropagation through time (BPTT) algorithm [17, 18, 19, 20, 21, 22, 23, 24, 25, 26]. While competitive accuracies are reported on the MNIST, CIFAR-10, and even ImageNet datasets [27, 28, 29], the gradient information is propagated each time step, whether or not a spike is emitted (as shown in Fig. 1). Therefore, these approaches do not follow the event-driven nature of spiking neural networks, which lose the asynchronous characteristic of SNNs and consume much power when trained on neuromorphic hardware.

The second category is event-driven algorithms, which propagate gradient information through spikes. Precise spiking timing acts an important role in this situation, and they are extensively used in such algorithms [30, 31, 32, 33, 34, 35, 36, 37]. Classical examples include SpikeProp [30] and its variants [31, 38, 39]. These algorithms approximate the derivative of spike timing to membrane potential as the negative inverse of the time derivative of membrane potential function. This approximation is actually mathematically correct without preconditions [40]. Some other works apply non-leaky integrate-and-fire neurons to stabilize the training process [33, 41, 36]. Most of these works restrict each neuron to fire at most once, which inspires [42] to take the spike time as the state of a neuron, and model the relation of neurons by this spike time. As a result, the SNN is trained similarly to an ANN. Among the methods allowing multiple spikes trained in an event-driven fashion, the state-of-the-art model is TSSL-BP [37]. However, they use RNN-like surrogate gradients (a sigmoid function) to assist training. It is still challenging to train SNNs in a pure event-driven fashion.

In this work, we develop a novel event-driven learning algorithm that can train high-performance deep SNNs. The main contributions of our work are as follows:

1. We prove that the typical SNN temporal backpropagation training approach assigns the gradient of an output spike of a neuron to the input spikes generating it. After summing this assignment rule altogether, we find that the sum of gradients is kept unchanged between layers.
2. We analyze the case of the pooling layer (which does not have weights) and find that average pooling does not keep the gradient sum unchanged, but we can modify its backward formulas to meet the requirement. Meanwhile, the max-pooling layer satisfies the rule initially.
3. We point out the reverse gradient problem in event-driven learning that the direction of the temporal gradient is reversed during backpropagation when the kernel function of an input spike is decreasing. Then we propose a backward kernel function that addresses this problem while keeping the sum of gradients unchanged between layers.
4. The adjusted average pooling layer and the non-decreasing backward kernel enhances the performance of our model as well as the convergence speed. To our best knowledge, our proposed approach achieves state-of-the-art performance on CIFAR10 among event-driven training methods (with temporal gradients) for SNNs. Meanwhile, our method is the first event-driven backpropagation approach that successfully trains SNN on the larger-scale CIFAR100 dataset.

2 Backgrounds and Related Work

The gradient-based learning of spiking neural networks contains two stages: the forward (inference) and the backward (learning) stages. In the forward stage, Leaky Integrate-and-Fire (LIF) neurons are most commonly used [20, 17, 25, 37], while other types of neurons are also applicable [30, 33]. Typically, these neuron models can be changed to the form of the Spike Response Model (SRM) [43, 44, 35], which is easily represented in an event-driven fashion.

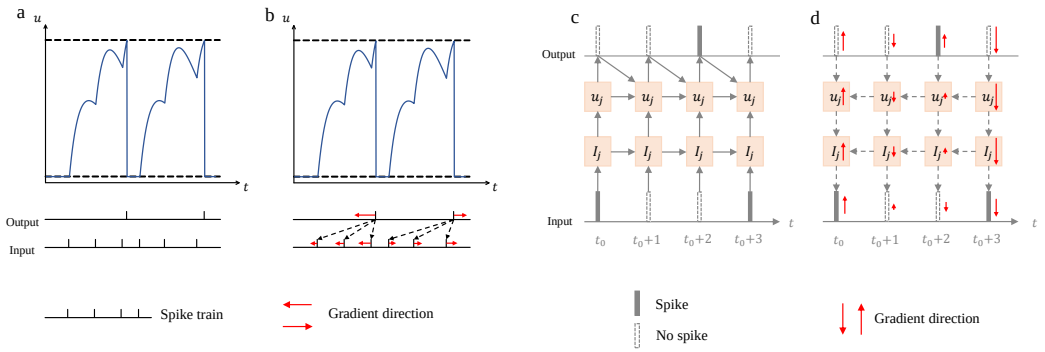

Figure 1: (a) Event-driven learning with (b) time-based gradient v.s. (c) RNN-like learning with (d) activation-based gradient.

In the backward stage, the situation is more complex. Here, we classify existing approaches from two dimensions: whether non-spike information is needed in discrete time steps (RNN-like) or not (event-driven) and whether the gradient represents spike scale (activation-based) or spike timing (time-based).

Event-driven learning v.s. RNN-like learning: In both forward and backward computation of event-driven learning, information is only carried by spikes in SNNs. Specifically, in backward computation, gradient information is propagated through spikes [30, 31, 39, 33] (shown in Fig. 1a-b). On the other side, in RNN-like learning, information is not only carried by spikes in backward computation. Especially, gradient information can be propagated through a neuron that does not emit a spike in backward computation (shown in Fig. 1c-d). This gradient propagation is achieved by a surrogate function [11, 16, 17, 22, 45], which is a function of the membrane potential at the current time step u_t, and the firing threshold θ.

Time-based gradient v.s. activation-based gradient: Time-based gradients represent the (reverse) direction that the timing of a spike should move, that is, to move leftward or rightward on the time axis [30]. In backward propagation, the derivative of the firing time of a spike to the corresponding membrane potential $\frac{\partial t}{\partial u}$ is often approximated as $\frac{-1}{\frac{\partial u}{\partial t}}$ [30, 31], denoting how the change of membrane potential will change the spike firing time (Fig. 1b). On the other side, activation-based approaches replace the Heaviside neuron activation function $\Theta(\cdot)$ ($s_t = \Theta(u_t - \theta)$) in forward propagation with derivable functions $\sigma(\cdot)$ in backward propagation, whether there are spikes in the current time step [17, 25, 29, 20]. Therefore, activation-based approaches essentially regard SNNs as binary RNNs and train them with approximated gradients, where the gradients indicate whether the values in the network (including the binary spikes) should be larger or smaller (Fig. 1d).

As a result, time-based gradients are event-driven by nature, since the temporal gradient could only be carried by spikes. Meanwhile, activation-based gradients are more suitable for the RNN-like training scheme since the diversity of surrogate gradients largely relies on the fact that $u_t \neq \theta$ in discrete time steps [16], which no longer holds in continuous time simulation. If we want to apply activation-based gradients to event-driven learning, there should only be one value $\frac{\partial s}{\partial u}$ when the membrane potential reaches the threshold.

Tab. 1 lists whether a gradient type can be used in a learning fashion. It should be noticed that although activation-based gradient is more suitable for RNN-like learning, it is still able to be used for event-driven learning.

Table 1: Whether the gradient type can be used in a learning fashion

	Event-driven learning	RNN-like learning
Time-based gradient	✓	×
Activation-based gradient	✓	✓

3 Methods

3.1 Forward Formulas

We use the spike response model [1] for neurons in the network. The forward propagation in the network can be described as follows:

$$u_i^{(l)}(t) = \int_{t_{i,last}^{(l)}}^{t} \left(\sum_j w_{ij}^{(l)} \cdot s_j^{(l-1)}(\tau) \right) \cdot \epsilon(t-\tau) d\tau, \tag{1}$$

$$s_i^{(l)}(t) = \delta(u_i^{(l)}(t) - \theta). \tag{2}$$

Here $u_i^{(l)}(t)$ denotes the membrane potential of neuron i in layer l at time t, $w_{ij}^{(l)}$ denotes the weight between neuron j in layer $l-1$ and neuron i in layer l. $t_{i,last}^{(l)}$ is the time of last spike of neuron i in layer l, and $s_i^{(l)}(t)$ represents the spike emitted from neuron i at time t. The function $\delta(\cdot)$ is the Dirac Delta function and θ is the firing threshold. The spike response kernel $\epsilon(t)$ can be described by

$$\epsilon(t) = \frac{\tau_m}{\tau_m - \tau_s}(e^{-\frac{t}{\tau_m}} - e^{-\frac{t}{\tau_s}}), \tag{3}$$

where τ_m and τ_s are the membrane time constant and the synapse time constant respectively. Notice that we do not use reset kernels as in previous works [39, 37]. Instead, we eliminate the influence of input spikes prior to the last output spike on membrane potentials.

3.2 Rethinking the Classical Time-based Backward Propagation Formula

In this subsection, we analyze the classical time-based backpropagation formula in SNNs. We first theoretically prove that it is essentially a weighted gradient assignment mechanism among layers. Then we check the pooling layer and show that the average pooling should be adjusted in backpropagation to satisfy the gradient assignment mechanism, while the max pooling naturally satisfies this mechanism.

Invariant sum of gradients among layers with weights. The most commonly used time-based gradient backpropagation method origins from [30]. The two key approximations are as follows:

$$\frac{\partial t_k(s_i^{(l)})}{\partial u_i^{(l)}(t_k)} = \frac{-1}{du_i^{(l)}(t_k)/dt} = \left(\sum_{t_{k,last}(s_i^{(l)}) < t_m(s_j^{(l-1)}) \leq t_k(s_i^{(l)})} w_{ij}^{(l)} \cdot \frac{\partial \epsilon(t_k - t_m)}{\partial t_m} \right)^{-1}, \tag{4}$$

$$\frac{\partial u_i^{(l)}(t_k)}{\partial t_m(s_j^{(l-1)})} = w_{ij}^{(l)} \cdot \frac{\partial \epsilon(t_k - t_m)}{\partial t_m}, \tag{5}$$

where $t_k(s_i^{(l)})$ denotes the firing time t_k of neuron i in layer l, $t_{k,last}(s_i^{(l)})$ is the firing time of the last spike emitted by neuron i before time t_k. $\frac{\partial t_k(s_i^{(l)})}{\partial \cdot}$ means the influence of changing other variables on the timing of a spike, and $\frac{\partial \cdot}{\partial t_k(s_i^{(l)})}$ is the influence of changing spike timing on that variable. Combining Eqs. 4-5 and the forward formulas, we can get an invariant equality:

$$\sum_{t_{k,last}(s_i^{(l)}) < t_m(s_j^{(l-1)}) \leq t_k(s_i^{(l)})} \frac{\partial t_k(s_i^{(l)})}{\partial t_m(s_j^{(l-1)})} = 1. \tag{6}$$

The proof is provided in Appendix. Eq. 6 implies the fact that the reference time ($t = 0$) is meaningless, and only relative spike times matter. If we increase all the spike times in layer $l-1$ by 1 unit along the time axis, then all the spike times in layer l are also increased by 1 unit along the time axis. Denote the loss function as L, then the gradient of L with respect to $t_m(s_j^{(l-1)})$ is:

$$\frac{\partial L}{\partial t_m(s_j^{(l-1)})} = \sum_i \sum_{t_m(s_j^{(l-1)}) < t_k(s_i^{(l)}) \leq t_{m,next}(s_j^{(l-1)})} \frac{\partial L}{\partial t_k(s_i^{(l)})} \cdot \frac{\partial t_k(s_i^{(l)})}{\partial t_m(s_j^{(l-1)})}, \tag{7}$$

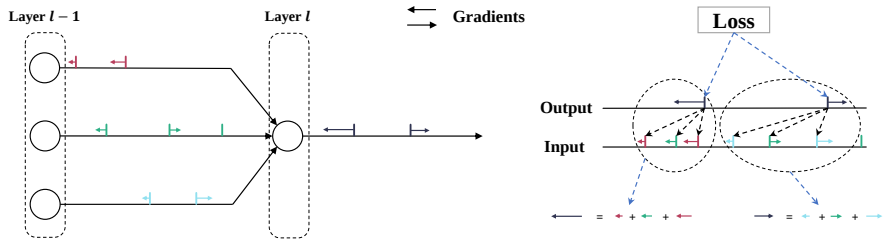

Figure 2: Left: Spikes (vertical ones) and their corresponding gradients (horizontal arrows) in layer $l-1$ and layer l. Right: Input and output of the neuron in layer l. The input integrates spikes from connected neurons in layer $l-1$ (notice the input is not multiplied by weights). In backpropagation, the gradients are transmitted from the output of the neuron to the inputs. The key property is that the gradient of an output spike **exactly equals** the sum of gradients of input spikes between the last output spike and it.

where $t_{m,next}(s_j^{(l-1)})$ denotes the firing time of the next spike emitted by neuron j after time t_m. Therefore, we actually decompose the gradient $\partial L/\partial t_k(s_i^{(l)})$ from layer l into (part of) a set of gradients $\partial L/\partial t_m(s_j^{(l-1)})$ of the last layer $l-1$, and keep their sum unchanged. In other words, we assign the weighted sum $\partial L/\partial t_k(s_i^{(l)})$ by weights $\partial t_k(s_i^{(l)})/\partial t_m(s_j^{(l-1)})$ to the gradients $\partial L/\partial t_m(s_j^{(l-1)})$ in the last layer, as shown in Fig. 2.

If we sum all the gradients together, we can get another invariant in this backpropagation rule:

$$\sum_i \sum_{t_k} \frac{\partial L}{\partial t_k(s_i^{(l)})} = \sum_j \sum_{t_m} \frac{\partial L}{\partial t_m(s_j^{(l-1)})}, \tag{8}$$

which means the sum of gradients $\sum_i \sum_{t_k} \frac{\partial L}{\partial t_k(s_i^{(l)})}$ never changes between layers under this rule.

Gradient sum invariance for pooling layers. The above function determines the gradient propagation in fully-connected and convolution layers. The case for pooling layers is illustrated in Fig. 3.

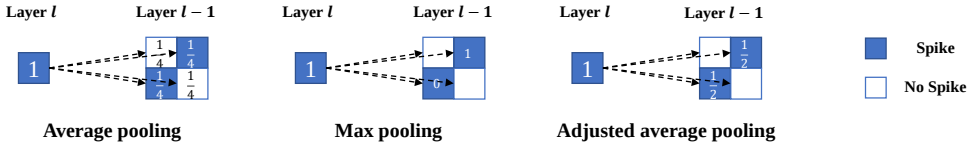

Figure 3: Different pooling strategies.

In average pooling with kernel size k, the gradient of one spike (at time t) in layer l is averagely propagated to $k \times k$ neurons in layer $l-1$ connected to it. Some of these $k \times k$ neurons may not emit spikes at time t. However, the gradients are also propagated to those neurons, which cannot further propagate the gradients to the previous layers. For instance, the two white squares (neurons) in layer $l-1$ in Fig. 3 receive gradients, but they will not further propagate the gradients to layer $l-2$. Thus, a part of the gradients is lost in the backpropagation of the average pooling layer, which might cause the gradient vanishing problem. Meanwhile, the sum of gradients carried by the spikes is not kept among layers in this case. We can adjust the backpropagation stage in average pooling to satisfy the gradient sum invariance requirement by increasing the multiplier from $1/k^2$ to $1/n_{spike}$, where n_{spike} is the number of spikes emitted by the $k \times k$ neurons in layer $l-1$ at the current time step.

On the contrary, in max pooling, the gradient of a spike in layer l is entirely propagated to one of the spikes emitted by its connected neurons in layer $l-1$. This maintains the property of the invariable sum of gradients.

It should be noticed that, although max pooling is different from adjusted average pooling in discrete simulation, they are almost surely the same in the continuous simulation since (almost surely) no two spikes emit at exactly the same time in this case.

3.3 Deficiencies of the Typical Time-based Propagation Formula and a New Approach

In this section, we first point out the reverse gradient problem in event-driven learning. Specifically, the gradient direction for spike timing gets wrong when the spike response kernel is decreasing. Then we propose a backward kernel that can not only solve the reverse gradient problem but also keep the property of the invariable sum of gradients.

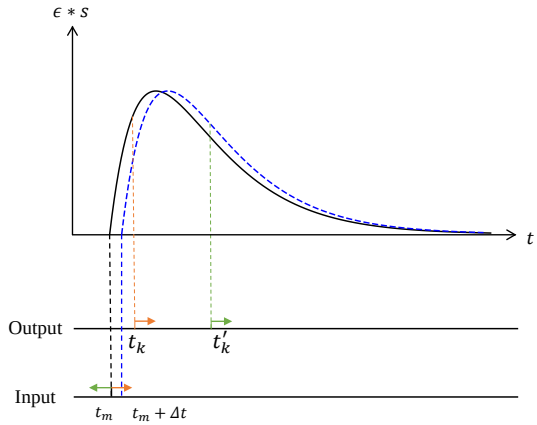

Figure 4: A comparison of temporal gradient direction in backpropagation between two different cases

The reverse gradient problem. Fig. 4 illustrates the membrane potential response of a neuron (with index i) to one of its input spikes (with a positive weight) from presynaptic neuron j, here we use the standard double-exponential spike response kernel (Eq. 3). Notice that this is not the whole membrane potential of the neuron i as it also receives the inputs from other presynaptic neurons. We consider two spike times (t_k and t'_k) of neuron i and one spike time t_m of presynaptic neuron j. If neuron i spikes at time t_k, moving the presynaptic spike from t_m to $t_m + \Delta t$ will decrease membrane potential at time t_k, which means postponing the spike at time t_k ($t_m \uparrow \Rightarrow u[t_k] \downarrow \Rightarrow t_k \uparrow$). This result shows that if the presynaptic neuron fires earlier, the postsynaptic neuron will fire earlier.

Oppositely, if the neuron spikes at t'_k, moving t_m to $t_m + \Delta t$ will cause a increase of $u[t'_k]$, which further moves t'_k leftward ($t_m \uparrow \Rightarrow u[t'_k] \uparrow \Rightarrow t'_k \downarrow$). As a result, if we want to move the output spike at t'_k leftward, we should move the input spike at t_m rightward, **which reverses the direction**. This might cause a problem: When we want to move t'_k leftward, we **want the neuron to emit more spikes**. However, in gradient backpropagation, it moves t_m rightward (assume weight $w > 0$), which may cause the neuron in the last layer to spike fewer, further **causing neurons in the current layer to spike fewer**.

More formally, we assume the neuron i receives a input spike at time t_m from presynaptic neuron j with synaptic weight w_{ij}, then the membrane potential of neuron i at time t_k is:

$$u_i(t_k) = w_{ij} \cdot \epsilon(t_k - t_m) + C, \tag{9}$$

where $\epsilon(t)$ denotes the spike response kernel (Eq. 3). C denotes the influence of other spikes, which is not in our concern here. In backward pass, according to Eqs. (4)-(5), we have

$$\frac{\partial L}{\partial t_m(s_j)} = \frac{\partial L}{\partial t_k(s_i)} \cdot \frac{\partial t_k(s_i)}{\partial u_i(t_k)} \cdot \frac{\partial u_i(t_k)}{\partial t_m(s_j)} = \frac{\partial L}{\partial t_k(s_i)} \cdot \frac{-1}{du_i(t_k)/dt} \cdot w_{ij} \cdot \frac{\partial \epsilon(t_k - t_m)}{\partial t_m}. \tag{10}$$

Note that when a spike is emitted by neuron i at time t_k, the slope of $u_i(t) > 0$ at time t_k, which means $\frac{-1}{du_i(t_k)/dt}$ has a negative sign. Considering $\frac{\partial \epsilon(t_k - t_m)}{\partial t_m} = -\frac{d\epsilon(\tau)}{d\tau}$, where $\tau = t_k - t_m$, we get:

$$\text{sign}\left(\frac{\partial L}{\partial t_m(s_j)}\right) = \text{sign}\left(\frac{\partial L}{\partial t_k(s_i)}\right) \cdot \text{sign}\left(w_{ij}\right) \cdot \text{sign}\left(\frac{d\epsilon(\tau)}{d\tau}\right). \tag{11}$$

When $\text{sign}\left(d\epsilon(\tau)/d\tau\right) = -1$, which is the part of the spike response kernel that decreases (see the case at t'_k in Fig. 4), the gradient direction of t_m can be classified into two cases: When $w_{ij} > 0$, $\text{sign}\left(\frac{\partial L}{\partial t_m(s_j)}\right) = -\text{sign}\left(\frac{\partial L}{\partial t_k(s_i)}\right)$, which means the gradient direction is reversed. When $w_{ij} < 0$, $\text{sign}\left(\frac{\partial L}{\partial t_m(s_j)}\right) = \text{sign}\left(\frac{\partial L}{\partial t_k(s_i)}\right)$, which means the gradient direction is kept.

In both cases, the sign of the gradient gets wrong in propagation between layers. Thus, the commonly used double-exponential spike response kernel is incompatible with the time-based gradient in event-driven learning.

A smoother gradient assigning approach. Inspired by the above gradient inconsistency as well as the invariance of gradient sum, we propose a new gradient backpropagation approach here. Specifically, we replace the function $\frac{\partial \epsilon(t_k - t_m)}{\partial t_m}$ in Eqs. (4) and (5) with a new function $h(t_k - t_m)$. Therefore, the backpropagation formula between layers turns into:

$$\begin{aligned}\frac{\partial t_k(s_i)}{\partial t_m(s_j)} &= \frac{\partial t_k(s_i)}{\partial u_i(t_k)} \cdot \frac{\partial u_i(t_k)}{\partial t_m(s_j)} \\ &= \begin{cases} 0, & \text{if} \quad t_m(s_j) \leq t_{k,last}(s_i) \text{ or } t_m(s_j) > t_k(s_i), \\ \left(\sum_{t_{k,last}(s_i) < t_m(s'_j) \leq t_k(s_i)} w_{ij} \cdot h(t_k - t_m)\right)^{-1} \cdot w_{ij} \cdot h(t_k - t_m), & \text{otherwise.} \end{cases}\end{aligned} \tag{12}$$

To guarantee that the gradients are not reversed between layers, we should expect $h(t) > 0$ always hold when $t > 0$. Besides, it can be see from Eq. 12 that $\frac{\partial t_k(s_i)}{\partial t_m(s_j)}$ will not change if we multiply $h(t)$ by an arbitrary constant, so we do not need to care about the scale of $h(t)$. Meanwhile, the property of invariable sum of gradients is kept after this replacement.

Therefore, we choose $h(t) = e^{-\frac{t}{\tau_{grad}}}$ to simplify the calculation, where τ_{grad} is a tunable parameter. Notice that the function $h(t)$ is only used in backward propagation, which means the spike response kernel in the forward propagation is not necessarily the integral of $h(t)$.

3.4 Overall Learning Rule

The loss function we use in this work is the counting loss function, which has the form $L = \frac{1}{N}\sum_{i=1}^{N_{out}} \left(\frac{1}{T}\left(N_i^{target} - \int_0^T s_i(t)dt\right)\right)^2$, where N_{out} is the number of output neurons and equals to the number of classes, $s_i(t)$ represents the spike train emitted by neuron i. Besides, N_i^{target} is the target of the spike number outputted by neuron i and typically we set N_i^{target} larger when i is the correct answer.

During the learning process, the gradient is first propagated from the loss function to the firing time of each spike from the last layer to the first layer. The formula for this stage is (please refer to Appendix for the detailed deduction):

$$\begin{aligned}\frac{\partial L}{\partial t_m(s_j^{(l-1)})} = \sum_i \frac{\partial L}{\partial t_{k,next}(s_i^{(l)})} \cdot \left(\sum_{t_i^{last}(s_i^{(l)}) < t_m(s_j^{(l-1)}) \leq t_{k,next}(s_i^{(l)})} w_{ij}^{(l)} \cdot h(t_{k,next} - t_m)\right)^{-1} \\ \cdot w_{ij}^{(l)} \cdot h(t_{k,next} - t_m),\end{aligned} \tag{13}$$

where $t_{k,next}(s_i^{(l)})$ is the firing time of the first spike emitted by neuron i after time t_m.

After this, the gradient to weights in each layer is calculated by summing up the multiplication of the gradients of spike firing times in the same layer and the derivative of weights with respect to spike firing times. The learning rule for this stage is

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \sum_{t_m(s_j^{(l-1)})} \frac{\partial L}{\partial t_{k,next}(s_i^{(l)})} \cdot \frac{-1}{\frac{\partial u_i^{(l)}(t_{k,next})}{\partial t}} \cdot \epsilon(t_{k,next} - t_m). \tag{14}$$

4 Experiments

In this section, we validate the effectiveness of our method on MNIST [46], Fashion-MNIST [47], N-MNIST [48], CIFAR10 [49], and CIFAR100 [49] datasets. We first introduce the training details,

then evaluate the performance of our algorithm and compare it with the state-of-the-art event-driven learning approaches. At last, we conduct ablation studies to illustrate the effectiveness of our proposed modules. More details of the configurations can be found in the Appendix.

4.1 Training Details

Initialization: When training in an event-driven fashion, gradient information is only carried by spikes. Therefore, the gradient information will be completely blocked by a layer when there are no spikes in that layer. To solve this problem, we start with layers of arbitrarily initialized weights and scale them by certain multiples, which can make the average firing rate to be a certain number for each layer. We obtain these multiple parameters by binary search and this strategy works well in practice.

Supervisory signals: Another problem we face is that neurons corresponding to certain classes do not fire anymore after certain training, which makes recognizing those classes correctly impossible in the following epochs. To address this problem, we utilize supervisory signals. For each neuron in the output layer corresponding to the ground-truth label, we force it to fire at the end of the simulation.

Experiment settings: In our experiments, we use the real-valued spike current representing the pixel intensities of the image as inputs. We list the network architectures each work uses in table 2. Notice that the output layer is, by default, a fully-connected layer containing the same number of neurons as the number of classes in the dataset, and omitted from the architecture representation. We run all experiments on a single Nvidia A100 GPU.

4.2 Comparison with the State-of-the-Art

Table 2: Performance comparison

Dataset	Model	Gradient Type	Architecture	Accuracy
MNIST	Mostafa [41]	Temporal	784-800	97.5%
	Cosma [32]	Temporal	784-340	97.9%
	S4NN [33]	Temporal	784-600	97.4%
	BS4NN [50]	Temporal	784-600	97.0%
	STiDi-BP [34]	Temporal	784-500	97.4%
	TSSL-BP [37]	Temporal	15C5-P2-40C5-P2-300[a]	99.53%
	STDBP [36]	Temporal	16C5-P2-32C5-P2-800-128	99.4%
	Ours	Temporal	15C5-P2-40C5-P2-300	99.47%
Fashion-MNIST	S4NN [33]	Temporal	784-1000	88.0%
	BS4NN [50]	Temporal	784-1000	87.3%
	STDBP [36]	Temporal	16C5-P2-32C5-P2-800-128	90.1%
	TSSL-BP [37]	Temporal	32C5-P2-64C5-P2-1024	92.83%
	Ours	Temporal	32C5-P2-64C5-P2-1024	93.28%
N-MNIST	TSSL-BP [37]	Temporal	12C5-P2-64C5-P2	99.40%
	Ours	Temporal	12C5-P2-64C5-P2	99.39%
CIFAR10	TSSL-BP [37]	Temporal	CIFARNet[b]	91.41%
	Ours	Temporal	VGG11[c]	91.88%
	Ours	Temporal	SEW-Resnet14	92.45%
CIFAR100	Ours	Temporal	VGG11	63.97%

[a] 15C5: convolution layer with 15 channels of 5×5 filters. P2: pooling layer with 2×2 filters.
[b] CIFARNet: 128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512
[c] VGG11: 128C3-128C3-P2-256C3-256C3-256C3-P2-512C3-512C3-512C3-P2-2048-2048

Tab. 2 reports the accuracies of the proposed method and other comparing methods. The performance of our algorithm is lower than TSSL-BP by 0.06% on the MNIST dataset and 0.01% on the N-MNIST dataset. However, the output of their network is real-valued postsynaptic currents while the output of our network is binary spikes. In addition, they use RNN-like gradients to assist learning. On the remaining datasets, we have achieved state-of-the-art performance among these works with temporal gradients. For the Fashion-MNIST dataset, our algorithm performs 0.45% higher than the previous SOTA. On the CIFAR10 dataset, we outperform the existing SOTA by 0.47%. For the CIFAR100

dataset, we are the first work to successfully train SNNs with time-based gradients in an event-driven fashion. We have achieved a performance of 63.97% on this dataset.

4.3 Ablation Studies

To show the effect of our proposed modules, we conduct ablation experiments on the CIFAR10 dataset. Specifically, two proposed components are taken into consideration: (1) As mentioned in Section 3.3, we compare the proposed gradient assignment functions $h(t) = e^{-\beta t}$ in (12) with the commonly used one $h(t) = \frac{d\epsilon(t)}{dt}$. (2) We compare the results of three different types of pooling layers (average pooling, adjusted average pooling and max pooling) mentioned in Section 3.2. We have tried all combinations of gradient assignment functions and pooling layers. The test accuracy of these different settings is shown in Tab. 3.

Table 3: Test accuracy comparison between different settings on CIFAR10

	Average pooling	Adjusted average pooling	Max pooling
$\frac{d\epsilon(t)}{dt}$	89.58%	91.52%	91.64%
$e^{-\beta t}$	89.80%	91.88%	91.79%

The results in Tab. 3 meets our expectation. For the pooling layer, max pooling and adjusted average pooling have much better performance than the average pooling. This accords with the conclusion in Section 3.2 that pooling layers keeping the property of invariant sum of gradients are better than those that do not.

The proposed gradient assignment function $h(t) = e^{-\beta t}$ is also better than the commonly used one $h(t) = \frac{d\epsilon(t)}{dt}$ for all three types of pooling layers. In addition, as shown in Fig. 5, $h(t) = e^{-\beta t}$ converges faster than $h(t) = \frac{d\epsilon(t)}{dt}$ in early stage.

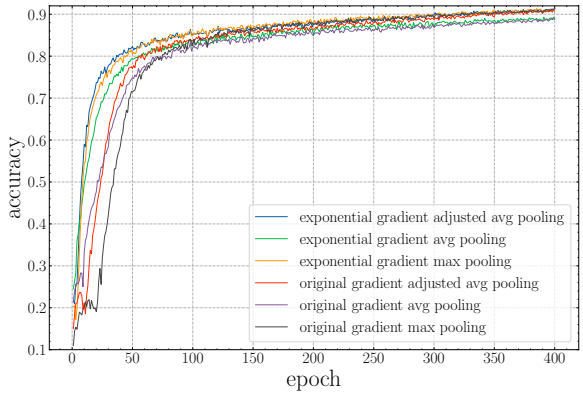

Figure 5: Test accuracy curve comparison between different settings on CIFAR10 (exponential gradient denotes $h(t) = e^{-\beta t}$ and original gradient denotes $h(t) = \frac{d\epsilon(t)}{dt}$).

5 Conclusion and Discussion

In this work, we analyze the commonly-used SNN temporal backpropagation training approach and find that it follows the gradient assignment rule. We also find the average pooling layer does not obey this rule while the max pooling layer does. We show that the direction of the temporal gradient will be reversed when the spike kernel is decreasing and avoid it with an increasing kernel in backpropagation. Our algorithm achieves state-of-the-art performance on CIFAR10 among time-based SNN learning approaches and successfully learns the parameters of SNN on CIFAR100 for the first time.

Compared with RNN-like methods, the proposed event-based learning algorithm has a lower computational cost and memory occupation when there are many time steps. Besides, our algorithm also does not need bias between layers. Meanwhile, gradient propagation between spikes instead of time steps can mitigate the gradient explosion/vanishing problem along the time axis.

However, there is still a gap between event-driven backpropagation and biological plausible learning, since event-driven backpropagation processes the spike train in reverse time, which conflicts with the online learning in the real world and desires for future research.

References

[1] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

[2] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[3] Youngeun Kim, Joshua Chough, and Priyadarshini Panda. Beyond classification: directly training spiking neural networks for semantic segmentation. *arXiv preprint arXiv:2110.07742*, 2021.

[4] Yeshwanth Venkatesha, Youngeun Kim, Leandros Tassiulas, and Priyadarshini Panda. Federated learning with spiking neural networks. *IEEE Transactions on Signal Processing*, 69:6183–6194, 2021.

[5] Youngeun Kim, Yeshwanth Venkatesha, and Priyadarshini Panda. Privatesnn: Privacy-preserving spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 1192–1200, 2022.

[6] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[7] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[8] De Ma, Juncheng Shen, Zonghua Gu, Ming Zhang, Xiaolei Zhu, Xiaoqiang Xu, Qi Xu, Yangjing Shen, and Gang Pan. Darwin: A neuromorphic hardware co-processor based on spiking neural networks. *Journal of Systems Architecture*, 77:43 – 51, 2017.

[9] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[10] Nicolas Perez-Nieves and Dan Goodman. Sparse spiking gradient descent. *Advances in Neural Information Processing Systems*, 34, 2021.

[11] Friedemann Zenke and Emre O Neftci. Brain-inspired learning on neuromorphic substrates. *Proceedings of the IEEE*, 109(5):935–950, 2021.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[13] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003.

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

[15] Friedemann Zenke, Sander M Bohté, Claudia Clopath, Iulia M Comşa, Julian Göltz, Wolfgang Maass, Timothée Masquelier, Richard Naud, Emre O Neftci, Mihai A Petrovici, et al. Visualizing a joint future of neuroscience and neuromorphic engineering. *Neuron*, 109(4):571–575, 2021.

[16] 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.

[17] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[18] Zenke F and Ganguli S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6), 2018.

[19] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in neuroscience*, page 119, 2020.

[20] 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.

[21] 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, 2021.

[22] Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in neuroscience*, page 1638, 2020.

[23] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International Conference on Learning Representations*, 2019.

[24] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[25] 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.

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

[27] 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*, 2021.

[28] 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, 2021.

[29] 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.

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

[31] Olaf Booij and Hieu tat Nguyen. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, 95(6):552–558, 2005.

[32] Iulia-Maria Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function: Learning with backpropagation. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2021.

[33] Saeed Reza Kheradpisheh and Timothée Masquelier. S4NN: Temporal backpropagation for spiking neural networks with one spike per neuron. *International Journal of Neural Systems*, 30(06):2050027, 2020.

[34] Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier. STiDi-BP: Spike time displacement based error backpropagation in multilayer spiking neural networks. *Neurocomputing*, 427:131–140, 2021.

[35] Timo C. Wunderlich and Christian Pehle. Event-based backpropagation can compute exact gradients for spiking neural networks. *Scientific Reports*, 11(1):12829, 2021.

[36] Malu Zhang, Jiadong Wang, Jibin Wu, Ammar Belatreche, Burin Amornpaisannon, Zhixuan Zhang, Venkata Pavan Kumar Miriyala, Hong Qu, Yansong Chua, Trevor E. Carlson, and Haizhou Li. Rectified linear postsynaptic potential function for backpropagation in deep spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2021.

[37] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, 2020.

[38] Chaofei Hong, Xile Wei, Jiang Wang, Bin Deng, Haitao Yu, and Yanqiu Che. Training spiking neural networks for cognitive tasks: A versatile framework compatible with various temporal codes. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4):1285–1296, 2019.

[39] Yan Xu, Xiaoqin Zeng, Lixin Han, and Jing Yang. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks. *Neural Networks*, 43:99–113, 2013.

[40] Jie Yang, Wenyu Yang, and Wei Wu. A remark on the error-backpropagation learning algorithm for spiking neural networks. *Applied Mathematics Letters*, 25(8):1118–1120, 2012.

[41] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3227–3235, 2017.

[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.

[43] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007.

[44] Jinseok Kim, Kyungsu Kim, and Jae-Joon Kim. Unifying activation-and timing-based learning rules for spiking neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 19534–19544, 2020.

[45] Youngeun Kim and Priyadarshini Panda. Optimizing deeper spiking neural networks for dynamic vision sensing. *Neural Networks*, 144:686–698, 2021.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[47] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[48] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015.

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

[50] Saeed Reza Kheradpisheh, Maryam Mirsadeghi, and Timothée Masquelier. BS4NN: Binarized spiking neural networks with temporal coding and learning. *Neural Processing Letters*, pages 1–19, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We use identical seeds to maximize the reproducibility.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The licence is accessible in the codes' homepage.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] No need. The datasets we used in this paper are public.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] No need. The datasets we used in this paper are public.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]