frontiers in Neuroscience

ORIGINAL RESEARCH
published: 23 May 2018
doi: 10.3389/fnins.2018.00331

Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks

Yujie Wu[1†], *Lei Deng*[1,2†], *Guoqi Li*[1†], *Jun Zhu*[3*] *and Luping Shi*[1*]

[1] *Department of Precision Instrument, Center for Brain-Inspired Computing Research, Beijing Innovation Center for Future Chip, Tsinghua University, Beijing, China,* [2] *Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States,* [3] *State Key Lab of Intelligence Technology and System, Tsinghua National Lab for Information Science and Technology, Tsinghua University, Beijing, China*

Spiking neural networks (SNNs) are promising in ascertaining brain-like behaviors since spikes are capable of encoding spatio-temporal information. Recent schemes, e.g., pre-training from artificial neural networks (ANNs) or direct training based on backpropagation (BP), make the high-performance supervised training of SNNs possible. However, these methods primarily fasten more attention on its spatial domain information, and the dynamics in temporal domain are attached less significance. Consequently, this might lead to the performance bottleneck, and scores of training techniques shall be additionally required. Another underlying problem is that the spike activity is naturally non-differentiable, raising more difficulties in supervised training of SNNs. In this paper, we propose a spatio-temporal backpropagation (STBP) algorithm for training high-performance SNNs. In order to solve the non-differentiable problem of SNNs, an approximated derivative for spike activity is proposed, being appropriate for gradient descent training. The STBP algorithm combines the layer-by-layer spatial domain (SD) and the timing-dependent temporal domain (TD), and does not require any additional complicated skill. We evaluate this method through adopting both the fully connected and convolutional architecture on the static MNIST dataset, a custom object detection dataset, and the dynamic N-MNIST dataset. Results bespeak that our approach achieves the best accuracy compared with existing state-of-the-art algorithms on spiking networks. This work provides a new perspective to investigate the high-performance SNNs for future brain-like computing paradigm with rich spatio-temporal dynamics.

Keywords: spiking neural network (SNN), spatio-temporal recognition, leaky integrate-and-fire neuron, MNIST-DVS, MNIST, backpropagation, convolutional neural networks (CNN)

OPEN ACCESS

Edited by:
Giacomo Indiveri,
Universität Zürich, Switzerland

Reviewed by:
Sadique Sheik,
University of California, San Diego, United States
Xianghong Lin,
Northwest Normal University, China

***Correspondence:**
Jun Zhu
dcszj@mail.tsinghua.edu.cn
Luping Shi
lpshi@tsinghua.edu.cn

[†] These authors have contributed equally to this work.

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *24 October 2017*
Accepted: *30 April 2018*
Published: *23 May 2018*

Citation:
Wu Y, Deng L, Li G, Zhu J and Shi L (2018) Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. Front. Neurosci. 12:331. doi: 10.3389/fnins.2018.00331

1. INTRODUCTION

Spiking neural network encodes information in virtue of the spike signals and shall be promising to effectuate more complicated cognitive functions in a way most approaching to the processing paradigm of brain cortex (Allen et al., 2009; Zhang et al., 2013; Kasabov and Capecci, 2015). SNNs are advantageous primarily due to the following two aspects: (1) more spatio-temporal information is encoded with spike pattern flows through SNNs, whereas most DNNs lack timing dynamics, especially the extensively-adopted feedforward DNNs; (2) more benefits can be achieved the hardware in virtue of the event-driven paradigm of SNNs, which has been leveraged by numerous

neuromorphic platforms (Benjamin et al., 2014; Furber et al., 2014; Merolla et al., 2014; Esser et al., 2016; Hwu et al., 2016; Zhang et al., 2016).

Yet the SNNs training still remains challenging because of the quite complicated dynamics and non-differentiable nature of the spike activity. In a nutshell, the existing training methods for SNNs fall into three types: (1) unsupervised learning; (2) indirect supervised learning; (3) direct supervised learning. The first one is originated from the weight modification of biological synapses, e.g., spike timing dependent plasticity (STDP) (Querlioz et al., 2013; Diehl and Cook, 2015; Kheradpisheh et al., 2016). Stemmed from its primary dependency on the local neuronal activities without global supervisor, effectuating high performance is quite difficult. The second one firstly trains an ANN, and thereupon transforms it into its SNN version with the same network structure where the rate of SNN neurons acts as the analog activity of ANN neurons (Peter et al., 2013; Hunsberger and Eliasmith, 2015; Neil et al., 2016). The last one is the direct supervised learning. Gradient descend, is a very popular optimization method for this learning type (Bohte et al., 2000; Schrauwen and Campenhout, 2004; Mckennoch et al., 2006; Lee et al., 2016). Spikeprop (Bohte et al., 2000; Schrauwen and Campenhout, 2004; Mckennoch et al., 2006) pioneered the gradient descent method to design multi-layer SNNs for supervised learning. It uses the first-spike time to encode input signals and minimizes the difference between the network output and desired signals, the whole process of which is similar to the traditional BP. Lee et al. (2016) treated the membrane potential as differentiable signals to solve the non-differential problems of spikes, and proposed a directly BP algorithm to train deep SNNs. Another efficient directly learning type is based on biological synaptic plasticity mechanism. Ponulak (2005); Ponulak and Kasiski (2010) developed the ReSuMe algorithm which uses STDP-like rule with remote supervision to learn the desired output spike sequence. Gtig and Sompolinsky (2006); Urbanczik and Senn (2009) proposed the tempotron learning rule which embeds information into spatio-temporal spike pattern and modifies synaptic connection by the output spike signals. Besides, some researchers utilized the unsupervised local plasticity mechanisms to abstract hierarchical features, and further modified network by parameters label signals (Mozafari et al., 2017; Tavanaei and Maida, 2017). Many emergent supervised training methods for SNNs have considered the spatial-temporal dynamic of spike-based neuron, but most of them primarily fasten more attention on one side of feature, either the spatial feature or the temporal feature, which in essence does not play out the advantage of SNNs and have to leverage several complicated skills to improve performance, such as error normalization, weight/threshold regularization, specific reset mechanism, etc. (Diehl et al., 2015; Lee et al., 2016; Neil et al., 2016). To this end, it is meaningful to design more general dynamic model and learning algorithm on SNNs.

In this paper, a direct supervised learning method is proposed for SNNs, combining both the spatial domain (SD) and temporal domain (TD) in the training phase. First and foremost, an iterative LIF model with SNNs dynamics is established which is appropriate for gradient descent training. On that basis, both the spatial dimension and temporal dimension are considered during the error backpropagation (BP) to evidently improves the network accuracy. An approximated derivative is introduced to address the non-differentiable issue of the spike activity. We test our SNNs model through adopting both the fully connected and convolutional architecture on the static MNIST dataset and a custom object detection dataset, as well as the dynamic N-MNIST dataset. Our method can make full use of spatio-temporal-domain (STD) information that captures the nature of SNNs, thus avoiding any complicated training skill. Experimental results indicate that the proposed method could achieve the best accuracy on both static and dynamic datasets compared with existing state-of-the-art algorithms. The influence of TD dynamics and different methods for the derivative approximation are analyzed systematically. This work enables to explore the high-performance SNNs for future brain-like computing paradigms with rich STD dynamics.

2. METHODS AND MATERIALS

We focus on how to efficiently train SNNs by taking full advantage of the spatio-temporal dynamics. In this section, we propose a learning algorithm that enables us to apply spatio-temporal BP for training spiking neural networks. To this end, subsection 2.1 firstly introduces an iterative leaky integrate-and-fire (LIF) model that are suitable for the error BP algorithm; subsection 2.2 gives the details of the proposed STBP algorithm; subsection 2.3 proposes the derivative approximation to address the non-differentiable issue.

2.1. Iterative Leaky Integrate-And-Fire Model in Spiking Neural Networks

It is known that Leaky Integrate-and-Fire (LIF) is the most commonly used model at present to describe the neuronal dynamics in SNNs, and it can be simply governed by:

$$\tau \frac{du(t)}{dt} = -u(t) + I(t) \tag{1}$$

where $u(t)$ is the neuronal membrane potential at time t, τ is a time constant and $I(t)$ denotes the pre-synaptic input which is determined by the pre-neuronal activities or external injections and the synaptic weights. When the membrane potential u exceeds a given threshold V_{th}, the neuron fires a spike and resets its potential to u_{reset}. As shown in **Figure 1**, the forward dataflow of the SNN propagates in the layer-by-layer SD like DNNs, and the self-feedback injection at each neuron node generates non-volatile integration in the TD. In this way, the whole SNNs run with complex STD dynamics and code spatio-temporal information into the spike pattern. The existing training algorithms only consider either the SD such as the supervised ones via BP, or the TD such as the unsupervised ones via timing-based plasticity, which might cause the performance bottleneck. Therefore, how to build a learning model by taking full use of the spatio-temporal domain (STD) forms the main motivation of this work.

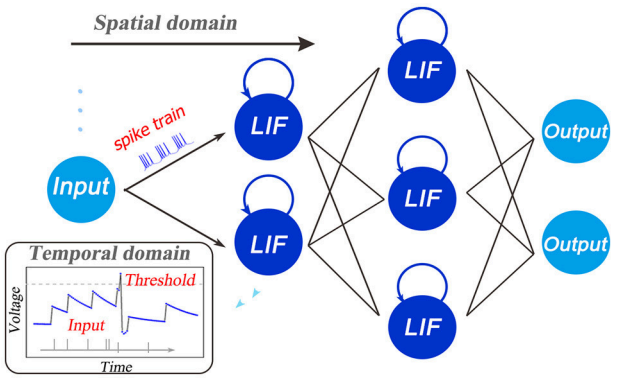

FIGURE 1 | Illustration of the spatio-temporal characteristic of SNNs. In addition to the layer-by-layer spatial dataflow like ANNs, SNNs are famous for the rich temporal dynamics. The existing training algorithms primarily fasten more attention on one side, either the spatial domain such as the supervised ones via backpropagation, or the temporal domain such as the unsupervised ones via timing-based plasticity. This causes the performance bottleneck. Therefore, how to build a framework for training high-performance SNNs by making full use of the STD information forms the major motivation of this work.

However, directly obtaining the analytic solution of LIF model in (1) makes it inconvenient to train SNNs based on BP with discrete dataflow. This is because the whole network presents complex dynamics in continuous TD. To address this issue, firstly we solve the linear differential Equation (1) with the initial condition $u(t)|_{t=t_{i-1}} = u(t_{i-1})$, and get the following iterative updating rule:

$$u(t) = u(t_{i-1})e^{\frac{t_{i-1}-t}{\tau}} + \hat{I}(t) \tag{2}$$

where the neuronal potential $u(t)$ in (1) depends on the previous potential at time t_{i-1} and the general pre-synaptic input $\hat{I}(t)$. The membrane potential exponentially decays until the neuron receives new input, and a new update round will start once the neuron fires a spike. That is to say, the neuronal state is co-determined by the spatial accumulations of $\hat{I}(t)$ and the leaky temporal memory of $u(t_{i-1})$.

As we know, the efficiency of error BP for training DNNs greatly benefits from the iterative representation of gradient descent which yields the chain rule for layer-by-layer error propagation in the SD backward pass. This motivates us to propose an iterative LIF based SNNS in which the iterations occur in both the SD and TD as follows:

$$x_i^{t+1,n} = \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1} \tag{3}$$

$$u_i^{t+1,n} = u_i^{t,n} f(o_i^{t,n}) + x_i^{t+1,n} + b_i^n \tag{4}$$

$$o_i^{t+1,n} = g(u_i^{t+1,n}) \tag{5}$$

where

$$f(x) = \tau e^{-\frac{x}{\tau}} \tag{6}$$

$$g(x) = \begin{cases} 1, & x \geq V_{th} \\ 0, & x < V_{th} \end{cases} \tag{7}$$

In above formulas, the upper index t denotes the time step t, and n and $l(n)$ denote the nth layer and the number of neurons in the nth layer, respectively. w_{ij} is the synaptic weight from the jth neuron in pre-synaptic layer to the ith neuron in the post-synaptic layer, and $o_j \in \{0, 1\}$ is the neuronal output of the jth neuron where $o_j = 1$ denotes a spike activity and $o_j = 0$ denotes nothing occurs. Equation (4) transforms Equation (2) to an iterative update of membrane potential in the LIF model. The first item on the right refers to the decay component of neuron potential corresponding to $u(t_{i-1})e^{\frac{t_{i-1}-t}{\tau}}$ in Equation (2), the second item x_i refers to the simplified representation of the pre-synaptic inputs of the ith neuron like $\hat{I}(t)$ in Equation (2), and the third item b_i is an equivalent variable to the fire threshold. Specifically, the threshold comparison of $u_i^{t+1,n} = u_i^{t,n} f(o_i^{t,n}) + x_i^{t+1,n} + b_i^n > V_{th}$ is equivalent to $u_i^{t+1,n} = u_i^{t,n} f(o_i^{t,n}) + x_i^{t+1,n} > V_{th} - b_i^n$, hence the modeling of adjustable bias b is utilized to mimic the threshold behavior.

Actually, formulas (4)–(5) are also inspired from the LSTM model (Hochreiter and Schmidhuber, 1997; Gers et al., 1999; Chung et al., 2015) by using a forget gate $f(.)$ to control the TD memory and an output gate $g(.)$ to fire a spike. The forget gate $f(.)$ controls the leaky extent of the potential memory in the TD, the output gate $g(.)$ generates a spike activity when it is activated. Specifically, for a small positive time constant τ, $f(.)$ can be approximated as:

$$f(o_i^{t,n}) \approx \begin{cases} \tau, & o_i^{t,n} = 0 \\ 0, & o_i^{t,n} = 1 \end{cases} \tag{8}$$

since $\tau e^{-\frac{1}{\tau}} \approx 0$. In this way, the LIF model could be transformed to an iterative version where the recursive relationship in both the SD and TD is clearly describe, which is suitable for the following gradient descent training in the STD.

2.2. Spatio-Temporal Backpropagation Training Framework

In order to present STBP training framework, we define the following loss function L in which the mean square error for all samples under a given time windows T is to be minimized:

$$L = \frac{1}{2S} \sum_{s=1}^{S} \| \boldsymbol{y_s} - \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{o_s}^{t,N} \|_2^2 \tag{9}$$

where $\boldsymbol{y_s}$ and $\boldsymbol{o_s}$ denote the label vector of the sth training sample and the neuronal output vector of the last layer N, respectively.

By combining Equations (3)–(9) together it can be seen that L is a function of $\boldsymbol{W}$ and $\boldsymbol{b}$. Thus, to obtain the derivative of L with respect to $\boldsymbol{W}$ and $\boldsymbol{b}$ is necessary for the gradient descent. Assume that we have obtained derivative of $\frac{\partial L}{\partial o_i}$ and $\frac{\partial L}{\partial u_i}$ at each layer n at time t, which is an essential step to obtain the final $\frac{\partial L}{\partial \boldsymbol{W}}$ and $\frac{\partial L}{\partial \boldsymbol{b}}$. **Figure 2** describes the error propagation (dependent on the derivation) in both the SD and TD at the single-neuron level

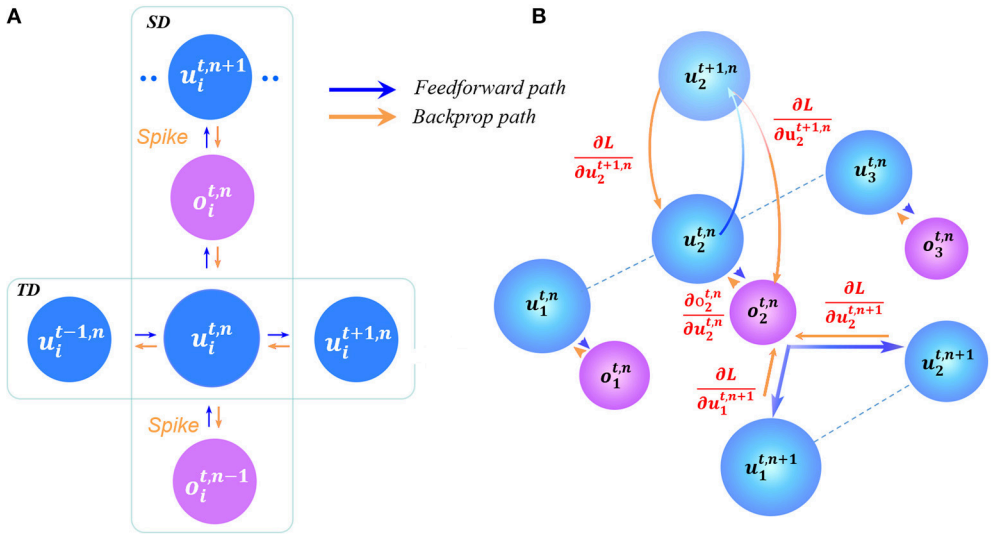

FIGURE 2 | Error propagation in the STD. **(A)** At the single-neuron level, the vertical path and horizontal path represent the error propagation in the SD and TD, respectively. **(B)** Similar propagation occurs at the network level, where the error in the SD requires the multiply-accumulate operation like the feedforward computation.

(**Figure 2A**) and the network level (**Figure 2B**). At the single-neuron level, the propagation is decomposed into a vertical path of SD and a horizontal path of TD. The dataflow of error propagation in the SD is similar to the typical BP for DNNs, i.e., each neuron accumulates the weighted error signals from the upper layer and iteratively updates the parameters in different layers. While the dataflow in the TD shares the same neuronal states, which makes it quite complicated to directly obtain the analytical solution. To solve this problem, we use the proposed iterative LIF model to unfold the state space in both the SD and TD direction, thus the states in the TD at different time steps can be distinguished, which enables the chain rule for iterative propagation. Similar idea can be found in the BPTT algorithm for training RNNs in Werbos (1990).

Now, we discuss how to obtain the complete gradient descent in the following four cases. Firstly, we denote that:

$$\delta_i^{t,n} = \frac{\partial L}{\partial o_i^{t,n}} \tag{10}$$

Case 1: $t = T$ **at the output layer** $n = N$.
In this case, the derivative $\frac{\partial L}{\partial o_i^{T,N}}$ can be directly obtained since it depends on the loss function in Equation (9) of the output layer. We could have:

$$\frac{\partial L}{\partial o_i^{T,N}} = -\frac{1}{TS}(y_i - \frac{1}{T}\sum_{k=1}^{T} o_i^{k,N}). \tag{11}$$

The derivation with respect to $u_i^{T,N}$ is generated based on $o_i^{T,N}$

$$\frac{\partial L}{\partial u_i^{T,N}} = \frac{\partial L}{\partial o_i^{T,N}}\frac{\partial o_i^{T,N}}{\partial u_i^{T,N}} = \delta_i^{T,N}\frac{\partial o_i^{T,N}}{\partial u_i^{T,N}}. \tag{12}$$

Case 2: $t = T$ **at the layers** $n < N$.
In this case, the derivative $\frac{\partial L}{\partial o_i^{T,n}}$ iteratively depends on the error propagation in the SD at time T as the typical BP algorithm. We have:

$$\frac{\partial L}{\partial o_i^{T,n}} = \sum_{j=1}^{l(n+1)} \delta_j^{T,n+1}\frac{\partial o_j^{T,n+1}}{\partial o_i^{T,n}} = \sum_{j=1}^{l(n+1)} \delta_j^{T,n+1}\frac{\partial g}{\partial u_i^{T,n}}w_{ji}. \tag{13}$$

Similarly, the derivative $\frac{\partial L}{\partial u_i^{T,n}}$ yields

$$\frac{\partial L}{\partial u_i^{T,n}} = \frac{\partial L}{\partial o_i^{T,n}}\frac{\partial o_i^{T,n}}{\partial u_i^{T,n}} = \delta_i^{T,n}\frac{\partial g}{\partial u_i^{T,n}}. \tag{14}$$

Case 3: $t < T$ **at the output layer** $n = N$.
In this case, the derivative $\frac{\partial L}{\partial o_i^{t,N}}$ depends on the error propagation in the TD direction. With the help of the proposed iterative LIF model in Equation (3)-(5) by unfolding the state space in the TD, we acquire the required derivative based on the chain rule in the TD as follows:

$$\begin{aligned}\frac{\partial L}{\partial o_i^{t,N}} &= \delta_i^{t+1,N}\frac{\partial o_i^{t+1,N}}{\partial o_i^{t,N}} + \frac{\partial L}{\partial o_i^{T,N}} \\ &= \delta_i^{t+1,N}\frac{\partial g}{\partial u_i^{t+1,N}}u_i^{t,N}\frac{\partial f}{\partial o_j^{t,N}} + \frac{\partial L}{\partial o_i^{T,N}},\end{aligned} \tag{15}$$

$$\frac{\partial L}{\partial u_i^{t,N}} = \frac{\partial L}{\partial u_i^{t+1,N}}\frac{\partial u_i^{t+1,N}}{\partial u_i^{t,N}} = \delta_i^{t+1,N}\frac{\partial g}{\partial u_i^{t+1,N}}f(o_i^{t,n}), \tag{16}$$

where $\frac{\partial L}{\partial o_i^{T,N}} = -\frac{1}{TS}(y_i - \frac{1}{T}\sum_{k=1}^{T} o_i^{k,N})$ as in Equation (11).

Case 4: $t < T$ **at the layers** $n < N$.
In this case, the derivative $\frac{\partial L}{\partial o_i^{t,n}}$ depends on the error propagation in both SD and TD. On one side, each neuron accumulates the weighted error signals from the upper layer in the SD like Case 2; on the other side, each neuron also receives the propagated error from self-feedback dynamics in the TD by iteratively unfolding the state space based on the chain rule like Case 3. So we have:

$$\frac{\partial L}{\partial o_i^{t,n}} = \sum_{j=1}^{l(n+1)} \delta_j^{t,n+1} \frac{\partial o_j^{t,n+1}}{\partial o_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial o_i^{t,n}} \tag{17}$$

$$= \sum_{j=1}^{l(n+1)} \delta_j^{t,n+1} \frac{\partial g}{\partial u_i^{t,n}} w_{ji} + \delta_i^{t+1,n} \frac{\partial g}{\partial u_i^{t,n}} u_i^{t,n} \frac{\partial f}{\partial o_i^{t,n}}, \tag{18}$$

$$\frac{\partial L}{\partial u_i^{t,n}} = \frac{\partial L}{\partial o_i^{t,n}} \frac{\partial o_i^{t,n}}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial u_i^{t,n}} \tag{19}$$

$$= \delta_i^{t,n} \frac{\partial g}{\partial u_i^{t,n}} + \delta_i^{t+1,n} \frac{\partial g}{\partial u_i^{t+1,n}} f(o_i^{t,n}). \tag{20}$$

Based on the four cases, the error propagation procedure (depending on the above derivatives) is shown in **Figure 2**. At the single-neuron level (**Figure 2A**), the propagation is decomposed into the vertical path of SD and the horizontal path of TD. At the network level (**Figure 2B**), the dataflow of error propagation in the SD is similar to the typical BP for DNNs, i.e. each neuron accumulates the weighted error signals from the upper layer and iteratively updates the parameters in different layers; and in the TD, the neuronal states are iteratively unfolded in the timing dimension that enables the chain-rule propagation. Finally, we obtain the derivatives with respect to $\boldsymbol{W}$ and $\boldsymbol{b}$ as follows:

$$\frac{\partial L}{\partial \boldsymbol{b}^{\boldsymbol{n}}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}} \frac{\partial \boldsymbol{u}^{\boldsymbol{t,n}}}{\partial L \boldsymbol{b}^{\boldsymbol{n}}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}}, \tag{21}$$

$$\frac{\partial L}{\partial \boldsymbol{W}^{\boldsymbol{n}}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}} \frac{\partial \boldsymbol{u}^{\boldsymbol{t,n}}}{\partial \boldsymbol{W}^{\boldsymbol{n}}}$$

$$= \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}} \frac{\partial \boldsymbol{u}^{\boldsymbol{t,n}}}{\partial \boldsymbol{x}^{\boldsymbol{t,n}}} \frac{\partial \boldsymbol{x}^{\boldsymbol{t,n}}}{\partial \boldsymbol{W}^{\boldsymbol{n}}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}} \boldsymbol{o}^{\boldsymbol{t,n-1}^T}, \tag{22}$$

where $\frac{\partial L}{\partial \boldsymbol{u}^{\boldsymbol{t,n}}}$ can be obtained from in Equation (11)–(17). Given the $\boldsymbol{W}$ and $\boldsymbol{b}$ according to the STBP, we can use gradient descent optimization algorithms to effectively train SNNs for achieving high performance.

2.3. Derivative Approximation of the Non-differentiable Spike Activity

In the previous sections, we have presented how to obtain the gradient information based on STBP, but the issue of non-differentiable points at each spiking time is yet to be addressed. Actually, the derivative of output gate $g(u)$ is required for the STBP training of Equation (11)–(21). Theoretically, $g(u)$ is a non-differentiable Dirac function of $\delta(u)$ which greatly challenges the effective learning of SNNs (Lee et al., 2016). $g(u)$ has zero value everywhere except an infinity value at zero, which causes the gradient vanishing or exploding issue that disables the error propagation. One of existing method views the discontinuous points of the potential at spiking times as noise and claimed it is beneficial for the model robustness (Bengio et al., 2015; Lee et al., 2016), while it did not directly address the non-differentiability of the spike activity. To this end, we introduce four curves to approximate the derivative of spike activity denoted by h_1, h_2, h_3, and h_4 in **Figure 3B**:

$$h_1(u) = \frac{1}{a_1} sign(|u - V_{th}| < \frac{a_1}{2}), \tag{23}$$

$$h_2(u) = (\frac{\sqrt{a_2}}{2} - \frac{a_2}{4}|u - V_{th}|) sign(\frac{2}{\sqrt{a_2}} - |u - V_{th}|), \tag{24}$$

$$h_3(u) = \frac{1}{a_3} \frac{e^{\frac{V_{th}-u}{a_3}}}{(1 + e^{\frac{V_{th}-u}{a_3}})^2}, \tag{25}$$

$$h_4(u) = \frac{1}{\sqrt{2\pi}\, a_4} e^{-\frac{(u-V_{th})^2}{2a_4}}, \tag{26}$$

where $a_i (i = 1, 2, 3, 4)$ determines the curve steepness, i.e., the peak width. In fact, h_1, h_2, h_3, and h_4 are the derivative of the rectangular function, polynomial function, sigmoid function and Gaussian cumulative distribution function, respectively. To be consistent with the Dirac function $\delta(u)$, we introduce the coefficient a_i to ensure the integral of each function is 1. Obviously, it can be proven that all the above candidates satisfy that:

$$\lim_{a_i \to 0^+} h_i(u) = \frac{dg}{du}, i = 1, 2, 3, 4. \tag{27}$$

Thus, $\frac{\partial g}{\partial u}$ in Equation (11)–(21) for STBP can be approximated by:

$$\frac{\partial g}{\partial u} \approx h_i(u), i = 1, 2, 3, 4. \tag{28}$$

In section 3.3, we will analyze the influence on the SNNs performance with different curves and different values of a_i.

3. RESULTS

3.1. Parameter Initialization

The initialization of parameters, such as the weights, thresholds and other parameters, is crucial for stabilizing the firing activities of the whole network. We should simultaneously ensure timely response of pre-synaptic stimulus but avoid too much spikes that reduces the neuronal selectivity. As it is known that the multiply-accumulate operations of the pre-spikes and weights, and the threshold comparison are two key computation steps in the forward pass. This indicates the relative magnitude between the weights and thresholds determines the effectiveness of parameter initialization. In this paper, we fix the threshold to be constant in each neuron for simplification, and only adjust the weights

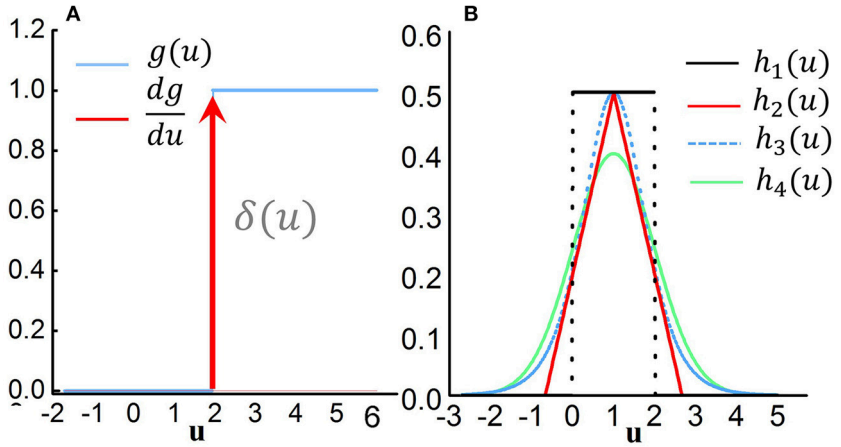

FIGURE 3 | Derivative approximation of the non-differentiable spike activity. **(A)** Step activation function of the spike activity and its original derivative function which is a typical Dirac function $\delta(u)$ with infinite value at $u = 0$ and zero value at other points. This non-differentiable property disables the error propagation. **(B)** Several typical curves to approximate the derivative of spike activity.

to control the activity balance. Firstly, we initial all the weight parameters by sampling from the standard uniform distribution:

$$\boldsymbol{W} \sim U[-1, 1] \tag{29}$$

Then, we normalize these parameters by:

$$w_{ij}^{n} = \frac{w_{ij}^{n}}{\sqrt{\sum_{j=1}^{l(n-1)} w_{ij}^{n2}}}, \quad i = 1, .., l(n) \tag{30}$$

The set of other parameters is presented in **Table 1**. Note that Adam (adaptive moment estimation Kingma and Ba, 2014) is a popular optimization method to accelerate the convergence speed of the gradient descent. When updating the parameters (W and b) based on their gradients (in Equation 21-22), we apply Adam optimizer that is usually used in ANNs. Actually, it does not affect the process of gradient acquire process, while just used for parameter update. The corresponding parameters are also listed in **Table 1**. Furthermore, throughout all the simulations in our work, any complex skill as in Diehl et al. (2015); Lee et al. (2016) is no longer required, such as the error normalization, weight/threshold regularization, fixed-amount-proportional reset mechanism, etc.

TABLE 1 | Parameters set in our experiments.

Network parameter	Description	Value
T	Time window	30 ms
V_{th}	Threshold (MNIST/object detection dataset/N-MNIST)	1.5, 2.0, 0.2
τ	Decay factor (MNIST/object detection dataset/N-MNIST)	0.1, 0.15, 0.2 ms
a_1, a_2, a_3, a_4	Derivative approximation parameters (**Figure 3**)	1.0
dt	Simulation time step	1 ms
r	Learning rate (SGD)	0.5
$\beta_1, \beta_2, \lambda$	Adam parameters	0.9, 0.999, 1-10^{-8}

3.2. Dataset Experiments

We test the STBP training framework on various datasets, including the static MNIST dataset, a custom object detection dataset as well as the dynamic N-MNIST dataset.

3.2.1. Spatio-Temporal Fully Connected Neural Network

3.2.1.1. Static dataset

The MNIST dataset of handwritten digits (Lecun et al., 1998) (**Figure 4B**) and a custom dataset for object detection (Zhang et al., 2016) (**Figure 4A**) are chosen to test our method. MNIST is comprised of a training set with 60,000 labeled hand-written digits, and a testing set of other 10,000 labeled digits, which are generated from the postal codes of 0-9. Each digit sample is a 28×28 grayscale image. The object detection dataset is a two-category image dataset created by our lab for pedestrian detection. It includes 1,509 training samples and 631 testing samples of 28×28 grayscale image. Actually, these images are patches in many real-world large-scale pictures, where each patch corresponds to an intrinsic location. The patches are sent to a neural network for binary classification to tell us whether or not an object exists in the scene, which is labeled by 0 or 1, as illustrated in **Figure 4A**. If so, an extra model will use the intrinsic location of this patch as the detected location. The input of the first layer should be a spike train, which requires us to convert the samples from the static datasets into spike events. To this end, the Bernoulli sampling conversion from original pixel intensity to the spike rate is used in this paper. Specifically, each normalized pixel is probabilistically converted to a spike event ("1") at each time step by using an independent and identically distributed Bernoulli sampling. The probability of generating a "1," i.e., a spike event, is proportional to the normalized value of the entry.

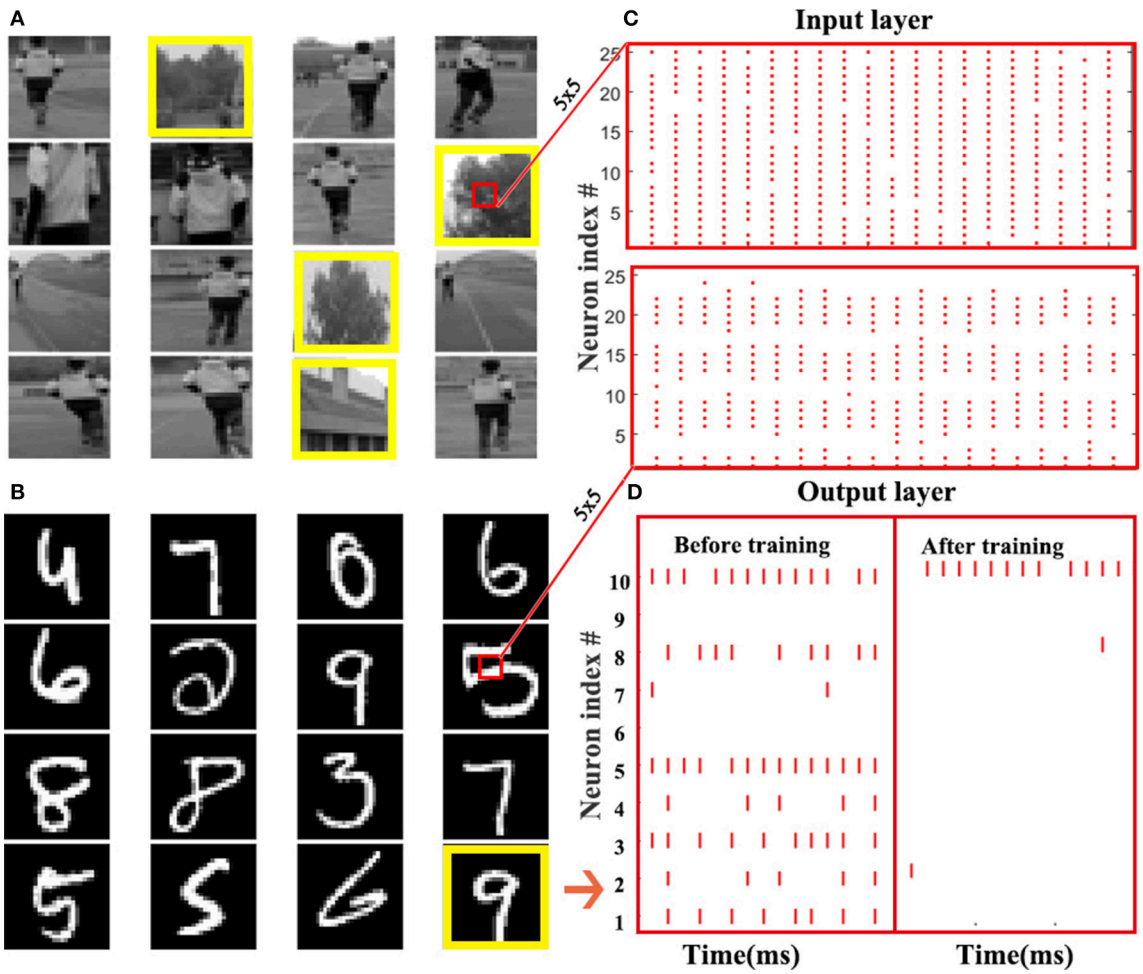

FIGURE 4 | Static dataset experiments. **(A)** A custom dataset for object detection. This dataset is a two-category image set built by our lab for pedestrian detection. By detecting whether there is a pedestrian, an image sample is labeled by 0 or 1. The images in the yellow boxes are labeled as 1, and the rest ones are marked as 0. **(B)** MNIST dataset. **(C)** Raster plot of the spike pattern converted from the center patch of 5×5 pixels of a sample in the object detection dataset (up) and MNIST (down). **(D)** Raster plot presents the comparison of output spike pattern over a digit 9 in MNIST dataset before and after the STBP training.

For example, if the pixel intensity is 0.8, it can generate a spike event at each time step with probability of 0.8 ant remain silent ("1") with probability of $0.2 = 1 - 0.8$. Then, the spike events within a certain time window form a spike train. The upper and lower sub-figures in **Figure 4C** are the spike pattern of 25 input neurons converted from the center patch of 5×5 pixels of a sample on the object detection dataset and MNIST, respectively. **Figure 4D** illustrates an example for the spike pattern of output layer within 15ms before and after the STBP training over the stimulus of digit 9. At the beginning, neurons in the output layer randomly fires, while after the training the 10*th* neuron coding digit 9 fires most intensively that indicates correct inference is achieved.

Table 2 compares our method with several other advanced results that uses the structure similar to Multi-layer Perceptron (MLP). Although we do not use any complex skill, the proposed STBP training method also outperforms all the reported results. We can achieve 98.89% testing accuracy which performs the best. **Table 3** compares our model with the typical MLP on the object detection dataset. The baseline model is one of the typical artificial neural networks (ANNs), i.e., not SNNs, and in the following we use 'non-spiking network' to distinguish them. It can be seen that our model achieves comparable performance with the non-spiking MLP. Note that the overall firing rate of the input spike train from the object detection dataset is higher than the one from MNIST dataset, so we increase its threshold to 2.0 in the simulation experiments.

3.2.1.2. Dynamic dataset

Compared with the static dataset, dynamic dataset, such as the N-MNIST (Orchard et al., 2015), contains richer temporal features, and therefore it is more suitable to exploit SNN's potential ability. We use the N-MNIST database as an example to evaluate the capability of our STBP method on dynamic dataset. N-MNIST converts the mentioned static MNIST dataset into its dynamic version of spike train by using the dynamic vision sensor

TABLE 2 | Comparison with the state-of-the-art spiking networks with similar architecture on MNIST.

Model	Network structure	Training skills	Accuracy%
Spiking RBM (STDP) (Neftci et al., 2013)	784-500-40	None	93.16
Spiking RBM(pre-training*) (Peter et al., 2013)	784-500-500-10	None	97.48
Spiking MLP(pre-training*) (Diehl et al., 2015)	784-1200-1200-10	Weight normalization	98.64
Spiking MLP(pre-training*) (Hunsberger and Eliasmith, 2015)	784-500-200-10	None	98.37
Spiking MLP(BP) (O'Connor and Welling, 2016)	784-200-200-10	None	97.66
Spiking MLP(STDP) (Diehl and Cook, 2015)	784-6400	None	95.00
Spiking MLP(BP) (Lee et al., 2016)	784-800-10	Error normalization/ parameter regularization	98.71
Spiking MLP(STBP)	784-800-10	None	**98.89**

*We mainly compare with these methods that have the similar network architecture, and * means that their model is based on pre-trained ANN models.*

TABLE 3 | Comparison with the typical MLP over object detection dataset.

Model	Network structure	Accuracy	
		Mean	Interval*
Non-spiking MLP(BP)	784-400-10	98.31%	[97.62%, 98.57%]
Spiking MLP(STBP)	784-400-10	**98.34%**	[**97.94%**, **98.57%**]

**Results with epochs [201,210].*

(DVS) (Lichtsteiner et al., 2007). For each original sample from MNIST, the work (Orchard et al., 2015) controls the DVS to move in the direction of three sides of the isosceles triangle in turn (**Figure 5B**) and collects the generated spike train which is triggered by the intensity change at each pixel. **Figure 5A** records the saccade results on digit 0. Each sub-graph records the spike train within 10ms and each 100ms represents one saccade period. Due to the two possible change directions of each pixel intensity (brighter or darker), DVS could capture the corresponding two kinds of spike events, denoted by on event and off event, respectively (**Figure 5C**). Since N-MNIST allows the relative shift of images during the saccade process, it produces 34×34 pixel range. And from the spatio-temporal representation in **Figure 5C**, we can see that the on-events and off-events are so different that we use two channel to distinguish it. Therefore, the network structure is $34\times34\times2$-400-400-10.

Table 4 compares our STBP method with some state-of-the-art results on N-MNIST dataset. The upper 5 results are based on ANNs, and lower 4 results including our method uses SNNs. The ANNs methods usually adopt a frame-based method, which collects the spike events in a time interval ($30 \sim 300ms$) to form a frame of image, and use the conventional algorithms for image classification to train the networks. Since the transformed images are often blurred, the frame-based preprocessing is harmful for model performance and abandons the hardware friendly event-driven paradigm. As can be seen from **Table 4**, the models of ANN are generally worsen than the models of SNNs.

In contrast, SNNs could naturally handle event stream patterns, and via better use of spatio-temporal features, our proposed STBP method achieves best accuracy of 98.78% when compared all the reported ANNs and SNNs methods. The greatest advantage of our method is that we did not use any complex training skill, which is beneficial for future hardware implementation.

3.2.2. Spatio-Temporal Convolution Neural Network

Extending our framework to convolution neural network structure allows the network going deeper and grants network more powerful SD information. Here we use our framework to establish the spatio-temporal convolution neural network. Compared with our spatio-temporal fully connected network, the main difference is the processing of the input image, where we use the convolution in place of the weighted summation. Specifically, in the convolution layer, each convolution neuron receives the convoluted results as input and updates its state according to the LIF model. In the pooling layer, because the binary coding of SNNs is inappropriate for standard max pooling, we use the average pooling instead.

Our spiking CNN model are tested on the MNIST dataset as well as the object detection dataset . In the MNIST, our network contains two convolution layers with kernel size of 5×5 and two average pooling layers alternatively, followed by one full connected layer. And like traditional CNN, we use the elastic distortion (Simard et al., 2003) to preprocess dataset. **Table 5** records the state-of-the-art performance of spiking convolution neural networks over MNIST dataset. Our proposed spiking CNN model obtain 98.42% accuracy, which outperforms other reported spiking networks with slightly lighter structure. Furthermore, we configure the same network structure on a custom object detection database to evaluate the proposed model performance. The testing accuracy is reported after training 200 epochs. **Table 6** indicates our spiking CNN model could achieve a competitive performance with the non-spiking CNN.

3.3. Performance Analysis

3.3.1. The Impact of Derivative Approximation Curves

In subsection 2.3 , we introduce different curves to approximate the ideal derivative of the spike activity. Here we try to analyze the influence of different approximation curves on the testing accuracy. The experiments are conducted on the MNIST dataset,

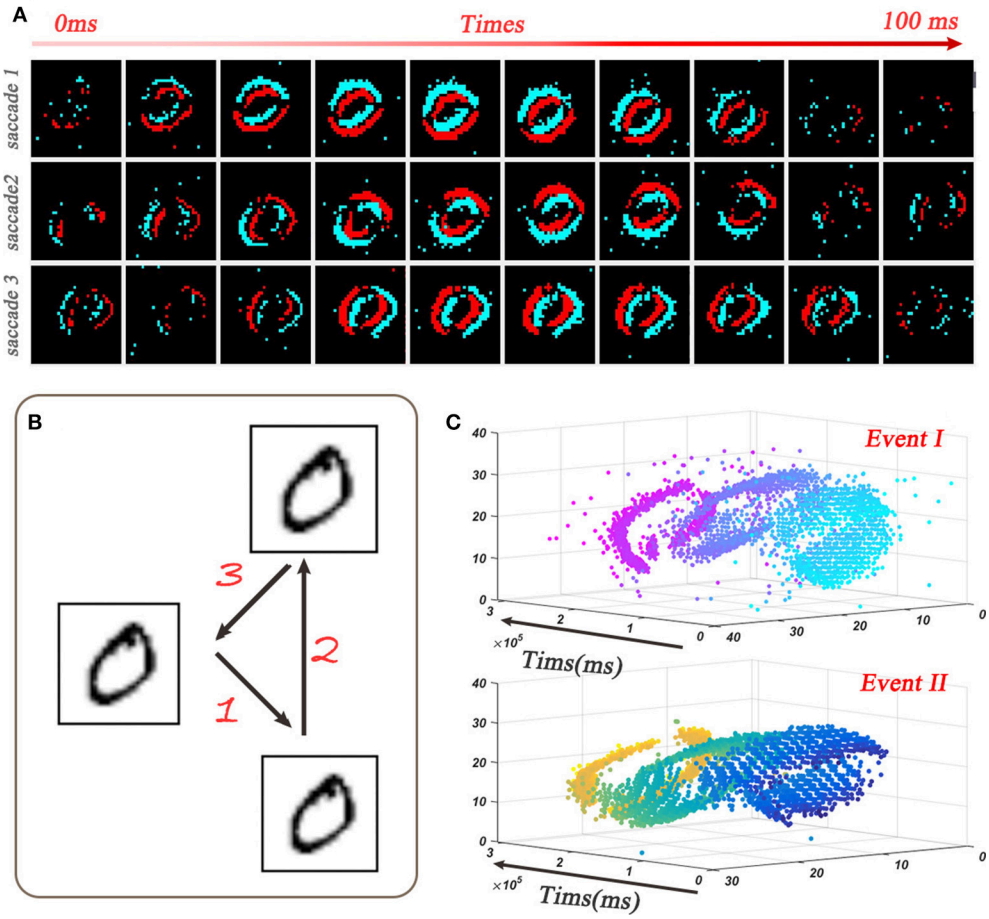

FIGURE 5 | Dynamic dataset of N-MNIST. **(A)** Each sub-picture shows a 10ms-width spike train during the saccades. **(B)** Spike train is generated by moving the dynamic vision sensor (DVS) in turn toward the direction of 1, 2, and 3. **(C)** Spatio-temporal representation of the spike train from digit 0 (Orchard et al., 2015)where the upper one and lower one denote the on-events and off-events, respectively.

TABLE 4 | Comparison with state-of-the-art networks over N-MNIST.

Model	Network structure	Training skills	Accuracy%
Non-spiking CNN(BP) (Neil et al., 2016)	-	None	95.30
Non-spiking CNN(BP) (Neil and Liu, 2016)	-	None	98.30
Non-spiking MLP(BP)(Lee et al., 2016)	34 × 34 × 2-800-10	None	97.80
LSTM(BPTT) (Neil et al., 2016)	-	Batch normalization	97.05
Phased-LSTM(BPTT) (Neil et al., 2016)	-	None	97.38
Spiking CNN(pre-training*) (Neil and Liu, 2016)	-	None	95.72
Spiking MLP(BP) (Lee et al., 2016)	34 × 34 × 2-800-10	Error normalization/parameter regularization	98.74
Spiking MLP(BP) (Cohen et al., 2016)	34 × 34 × 2-10000-10	None	92.87
Spiking MLP(STBP)	34 × 34 × 2-800-10	None	**98.78**

*We only show the network structure based on MLP, and the other network structure refers to the above references. *means that their model is based on pre-trained ANN models.*

and the network structure is $784 - 400 - 10$. The testing accuracy is reported after training 200 epochs. Firstly, we compare the impact of different curve shapes on model performance. In our simulation we use the mentioned h_1, h_2, h_3, and h_4 shown in **Figure 3B**. **Figure 6A** illustrates the results of approximations of different shapes. We observe that different nonlinear curves, such as h_1, h_2, h_3, and h_4, only present small variations on the performance.

Furthermore, we use the rectangular approximation as an example to explore the impact of curve steepness (or peck width)

on the experiment results. We set $a_1 = 0.1, 1.0, 2.5, 5.0, 7.5, 10$ and the corresponding results are plotted in **Figure 6B**. Different colors denote different a_1 values. Actually, a_1 in the rang of 0.5–5.0 achieves comparable convergence while too large ($a_1 = 10$) or too small ($a_1 = 0.1$) value performs worse performance. Combining **Figures 6A,B**, it indicates that the key point for approximating the derivation of the spike activity is to capture the nonlinear nature and proper curve steepness, while the specific curve shape is not so critical.

TABLE 5 | Comparison with other spiking CNN over MNIST.

Model	Network structure	Accuracy
Spiking CNN (pre-training*) (Esser et al., 2016)	28×28×1-12C5-P2-64C5-P2-10	99.12%
Spiking CNN(BP) (Lee et al., 2016)	28×28×1-20C5-P2-50C5-P2-200-10	99.31%
Spiking CNN (STBP)	28×28×1-15C5-P2-40C5-P2-300-10	**99.42%**

*We mainly compare with these methods that have the similar network architecture, and *means that their model is based on pre-trained ANN models.*

TABLE 6 | Comparison with the typical CNN over object detection dataset.

Model	Network structure	Accuracy	
		Mean	Interval*
Non-spiking CNN(BP)	28 × 28 × 1-6C3-300-10	98.57%	[98.57%, 98.57%]
Spiking CNN(STBP)	28 × 28 × 1-6C3-300-10	**98.59%**	[**98.26%**, **98.89%**]

**Results with epochs [201,210].*

3.3.2. The Impact of Temporal Domain

A major contribution of this work is introducing the temporal domain into the existing spatial domain based BP training method, which makes full use of the spatio-temporal dynamics of SNNs and enables the high-performance training. Now we quantitatively analyze the impact of the TD item. The experiment configurations keep the same with the previous section ($784 - 400 - 10$) and we also report the testing results after training 200 epochs. Here the existing BP in the SD is termed as SDBP.

Table 7 records the simulation results. The testing accuracy of SDBP is lower than the accuracy of the STBP on different datasets, which shows the temporal information is beneficial for model performance. Specifically, compared to the STBP, the SDBP has a 1.21% loss of accuracy on the objective recognition dataset, which is 5 times larger than the loss on the MNIST. And results also imply that the performance of SDBP is not stable enough. In addition to the interference of the dataset itself, the reason for this variation may be the unstability of

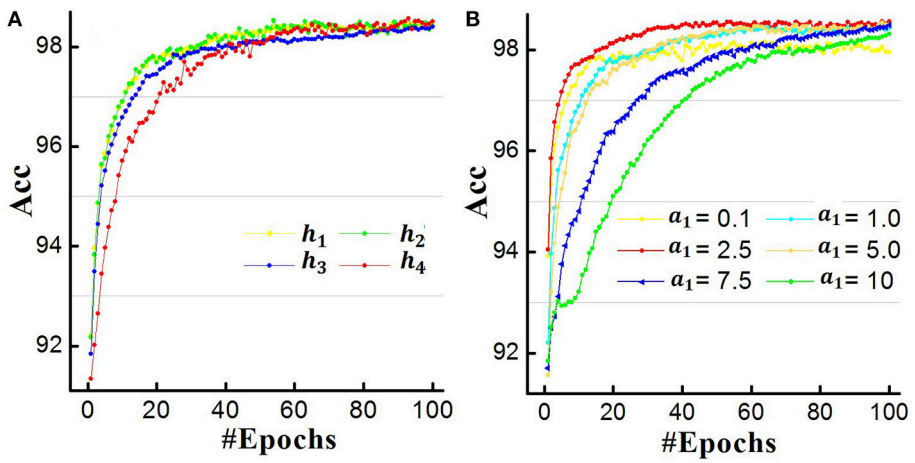

FIGURE 6 | Comparisons of different derivation approximation curves. **(A)** The influence of curve shape. **(B)** The influence of curve steepness/width.

TABLE 7 | Comparison for the SDBP model and the STBP model on different datasets.

Model	Dataset	Network structure	Training skills	Accuracy	
				Mean	Interval*
Spiking MLP	Objective recognition	784-400-10	None	97.11%	[96.04%,97.78%]
(SDBP)	MNIST	784-400-10	None	98.29%	[98.23%, 98.39%]
Spiking MLP	Objective recognition	784-400-10	None	**98.32%**	[**97.94%**, **98.57%**]
(STBP)	MNIST	784-400-10	None	**98.48%**	[**98.42%**, **98.51%**]

**Results with epochs [201,210].*

SNNs training method. Actually, the training of SNNs relies heavily on the parameter initialization, which is also a great challenge for SNNs applications. In many reported works, researchers usually leverage some special skills or mechanisms to improve the training performance, such as the regularization, normalization, etc. In contrast, by using our STBP training method, much higher performance can be achieved on the same network (98.48% on MNIST and 98.32% on the object detection dataset). Note that the STBP didn't use any complex training skill. This stability and robustness indicate that the dynamics in the TD fundamentally includes great potential for the SNNs computing and this work indeed provides an insightful evidence.

4. DISCUSSION

In this work, we propose a spatio-temporal backpropagation (STBP) algorithm that allows to effective supervised learning for SNNs. Although existing supervised learning methods have considered either SD feature or TD feature (Gtig and Sompolinsky, 2006; Lee et al., 2016; O'Connor and Welling, 2016), they do not combine them well, which may cause their model hardly to get high-accuracy results on some standard benchmarks. Although indirect training methods (Diehl et al., 2015; Hunsberger and Eliasmith, 2015) achieve performance very close to the pre-trained model, the conversion strategy essentially helps little to understand the nature of SNNs. By combining the information in both SD and TD domain, our STBP algorithm can bridge this gap. We implement STBP on both MLP and CNN architecture, which are verified on both static and dynamic datasets. Results of our model are superior to the existing state-of-the-art SNNs on relatively small-scale networks of spiking MLP and CNNs, and even outperforms non-spiking DNNs with the same network size on dynamic N-MNIST dataset. Specifically, on MNIST, we achieve 98.89% accuracy with fully connected architecture and achieve 99.42% accuracy with convolutional architecture. On N-MNIST, we achieve 98.78% accuracy with fully connected architecture, to the best of our knowledge, which beats previous works on this dataset.

Furthermore, we introduce an approximated derivative to address the non-differentiable issue of the spike activity. Previous works regard the non-differentiable points as noise (Vincent et al., 2008; Hunsberger and Eliasmith, 2015), while our results reveal that the steepness and width of the approximation curve would affect the learning performance, while the specific curve shape is not so critical. Another attractive advantage of our algorithm is that it does not need complex training skills which are widely used in existing schemes to guarantee the performance (Diehl et al., 2015; Lee et al., 2016; Neil et al., 2016), that makes it easier to be implemented in large-scale networks. These results also indicate that the use of spatio-temporal complexity to solve problems captures one of the key potentials of SNNs. Because the brain leverages complexity in both the temporal and spatial domain to solve problems, we also would like to claim that implementing the STBP on SNNs is more bio-plausible than applying the spatial BP like that in DNNs. The remove of extra training skills also makes it more hardware-friendly for the design of neuromorphic chips with online learning ability.

Since the N-MNIST converts the static MNIST into a dynamic event-driven version by the relative movement of DVS, in essence this generation method could not provide sufficient temporal information and additional data feature than original database. Hence it is important to further apply our model to tackle more convincing problems with temporal characteristics, such as TIMIT (Garofolo et al., 1993), Spoken Digits database (Davis et al., 1952).

We also evaluate our model on CIFAR-10 dataset. Here we do not resort to any data argument methods and training techniques (e.g., batch normalization, weight decay). Considering the training speed, we adopt a small-scale structure with 2 convolution layers (20 channels with kernel 5×5 - 30 channels 5×5), 2×2 average-pooling layers after each convolution layer, followed by 2 fully connected layers (256 and 10 neurons, respectively). Testing accuracy is reported after 100 training epochs. The spiking CNN achieves 50.7% accuracy and the ANN with same structure achieves 52.9% accuracy. It suggests that SNN is able to obtain comparable performance on larger datasets. To the best of our knowledge, currently few works report the results on CIFAR10 for direct training of SNNs (not including those pre-trained ANN models). The difficulty of this problem mainly involves two aspects. Firstly, it is challenging to implement BP algorithm to train SNNs directly at this stage because of the complex dynamics and non-differentiable spike activity . Secondly, although it is energy efficient to realize SNN on specialized neuromorphic chips, it is very difficult and time-consuming to simulate the complex kinetic behaviors of SNN on computer software (about ten times or even hundred times the runtimes compared to the same structure ANN). Therefore, accelerating the supervised training of large scale SNNs based on CPU/GPU or neuromorphic substrates is also worth studying in the future.

AUTHOR CONTRIBUTIONS

YW and LD proposed the idea, designed and did the experiments. YW, LD, GL, and JZ conducted the modeling work. YW, LD, and GL wrote the manuscript, then JZ and LS revised it. LS directed the projects and provided overall guidance.

ACKNOWLEDGMENTS

The work was partially supported by National Natural Science Foundation of China (61603209), the Study of Brain-Inspired Computing System of Tsinghua University program (20151080467), Beijing Natural Science Foundation (4164086), Independent Research Plan of Tsinghua University (20151080467), and by the Science and Technology Plan of Beijing, China (Grant No. Z151100000915071).

REFERENCES

Allen, J. N., Abdel-Aty-Zohdy, H. S., and Ewing, R. L. (2009). "Cognitive processing using spiking neural networks," in *IEEE 2009 National Aerospace and Electronics Conference* (Dayton, OH), 56–64.

Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., and Wu, Y. (2015). An objective function for stdp. *Comput. Sci. preprint arXiv*.

Benjamin, B. V., Gao, P., Mcquinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Bohte, S. M., Kok, J. N., and Poutr, J. A. L. (2000). "Spikeprop: backpropagation for networks of spiking neurons," in *Esann 2000, European Symposium on Artificial Neural Networks, Bruges, Belgium, April 26-28, 2000, Proceedings* (Bruges), 419–424.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. *Comput. Sci.* 2067–2075. *preprint arXiv*.

Cohen, G. K., Orchard, G., Leng, S. H., Tapson, J., Benosman, R. B., and Schaik, A. V. (2016). Skimming digits: neuromorphic classification of spike-encoded images. *Front. Neurosci.* 10:184. doi: 10.3389/fnins.2016.00184

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *J. Acoust. Soc. Am.* 24:637. doi: 10.1121/1.1906946

Diehl, P. U., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Diehl, P. U., Neil, D., Binas, J., and Cook, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing" in *International Joint Conference on Neural Networks* (Killarney), 1–8.

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., Dahlgren, N. L. (1993). *DARPA TIMIT Acoustic-Phonetic Continous Speech Corpus CD-ROM. NIST Speech Disc 1-1.1.* NASA STI/Recon Technical Report n 93.

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: continual prediction with lstm. *Neural Comput.* 12, 2451–2571. doi: 10.1162/089976600300015015

Gtig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* 9:420–428. doi: 10.1038/nn1643

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.* 9, 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Hunsberger, E., and Eliasmith, C. (2015). Spiking deep networks with lif neurons. *Comput. Sci. preprint arXiv*.

Hwu, T., Isbell, J., Oros, N., and Krichmar, J. (2016). A self-driving robot using deep convolutional neural networks on neuromorphic hardware. *arXiv.org*.

Kasabov, N., and Capecci, E. (2015). Spiking neural network methodology for modelling, classification and understanding of eeg spatio-temporal data measuring cognitive processes. *Infor. Sci.* 294, 565–575. doi: 10.1016/j.ins.2014.06.028

Kheradpisheh, S. R., Ganjtabesh, M., and Masquelier, T. (2016). Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *Neurocomputing* 205, 382–392. doi: 10.1016/j.neucom.2016.04.029

Kingma, D., and Ba, J. (2015). "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Lichtsteiner, P., Posch, C., and Delbruck, T. (2007). A 128x128 120db 15us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circ.* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Mckennoch, S., Liu, D., and Bushnell, L. G. (2006). "Fast modifications of the spikeprop algorithm," in *IEEE International Joint Conference on Neural Network Proceedings* (Vancouver, BC), 3970–3977.

Merolla, P. A., Arthur, J. V., Alvarezicaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). Artificial brains. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzaridalini, A., and Ganjtabesh, M. (2017). First-spike based visual categorization using reward-modulated stdp. *preprint arXiv*.

Neftci, E., Das, S., Pedroni, B., Kreutzdelgado, K., and Cauwenberghs, G. (2013). Event-driven contrastive divergence for spiking neuromorphic systems. *Front. Neurosci.* 7:272. doi: 10.3389/fnins.2013.00272

Neil, D., and Liu, S. C. (2016). "Effective sensor fusion with event-based sensors and deep network architectures," in *IEEE International Symposium on Circuits and Systems*, ed O. René Levesque (Montréal, QC).

Neil, D., Pfeiffer, M., and Liu, S. C. (2016). Phased lstm: accelerating recurrent network training for long or event-based sequences. *arXiv.org*.

O'Connor, P., and Welling, M. (2016). Deep spiking networks. *arXiv.org*.

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437

Peter, O., Daniel, N., Liu, S. C., Tobi, D., and Michael, P. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosci.* 7:178. doi: 10.3389/fnins.2013.00178

Ponulak, F., (2005). *ReSuMe-New Supervised Learning Method for Spiking Neural Networks[J]*. Institute of Control and Information Engineering, Poznan University of Technology.

Ponulak, F., and Kasiski, A. (2010). Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting. *Neural Comput.* 22, 467–510. doi: 10.1162/neco.2009.11-08-901

Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Trans. Nanotechnol.* 12, 288–295. doi: 10.1109/TNANO.2013.2250995

Schrauwen, B., and Campenhout, J. V. (2004). Extending spikeprop. *arXiv.org* 1, 471–475.

Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). "Best practices for convolutional neural networks applied to visual document analysis," in *International Conference on Document Analysis and Recognition* (Edinburgh), 958.

Tavanaei, A., and Maida, A. S. (2017). Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning. *preprint arXiv*.

Urbanczik, R., and Senn, W. (2009). A gradient learning rule for the tempotron. *Neural Comput.* 21, 340–352. doi: 10.1162/neco.2008.09-07-605

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. A. (2008). "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine Learning* (Helsinki), 1096–1103.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 1550–1560. doi: 10.1109/5.58337

Zhang, B., Shi, L., and Song, S. (2016). Creating more intelligent robots through brain-inspired computing. *Science* 354:1445. doi: 10.1126/science.354.6318.1445-b

Zhang, X., Xu, Z., Henriquez, C., and Ferrari, S. (2013). "Spike-based indirect training of a spiking neural network-controlled virtual insect," in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on* (Florence: IEEE), 6798–6805.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.