Online Training Through Time for Spiking Neural Networks

Mingqing Xiao[1], **Qingyan Meng**[2,3], **Zongpeng Zhang**[4], **Di He**[1], **Zhouchen Lin**[1,5,6*]
[1]Key Lab. of Machine Perception (MoE), School of Intelligence Science and Technology, Peking University
[2]The Chinese University of Hong Kong, Shenzhen
[3]Shenzhen Research Institute of Big Data
[4]Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University
[5]Institute for Artificial Intelligence, Peking University
[6]Peng Cheng Laboratory, China
{mingqing_xiao, dihe, zlin}@pku.edu.cn, qingyanmeng@link.cuhk.edu.cn, zongpeng.zhang98@gmail.com

Abstract

Spiking neural networks (SNNs) are promising brain-inspired energy-efficient models. Recent progress in training methods has enabled successful deep SNNs on large-scale tasks with low latency. Particularly, backpropagation through time (BPTT) with surrogate gradients (SG) is popularly used to enable models to achieve high performance in a very small number of time steps. However, it is at the cost of large memory consumption for training, lack of theoretical clarity for optimization, and inconsistency with the online property of biological learning rules and rules on neuromorphic hardware. Other works connect the spike representations of SNNs with equivalent artificial neural network formulation and train SNNs by gradients from equivalent mappings to ensure descent directions. But they fail to achieve low latency and are also not online. In this work, we propose online training through time (OTTT) for SNNs, which is derived from BPTT to enable forward-in-time learning by tracking presynaptic activities and leveraging instantaneous loss and gradients. Meanwhile, we theoretically analyze and prove that the gradients of OTTT can provide a similar descent direction for optimization as gradients from equivalent mapping between spike representations under both feedforward and recurrent conditions. OTTT only requires constant training memory costs agnostic to time steps, avoiding the significant memory costs of BPTT for GPU training. Furthermore, the update rule of OTTT is in the form of three-factor Hebbian learning, which could pave a path for online on-chip learning. With OTTT, it is the first time that the two mainstream supervised SNN training methods, BPTT with SG and spike representation-based training, are connected, and meanwhile it is in a biologically plausible form. Experiments on CIFAR-10, CIFAR-100, ImageNet, and CIFAR10-DVS demonstrate the superior performance of our method on large-scale static and neuromorphic datasets in a small number of time steps. Our code is available at `https://github.com/pkuxmq/OTTT-SNN`.

1 Introduction

Spiking neural networks (SNNs) are regarded as the third generation of neural network models [1] and have gained increasing attention in recent years [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. SNNs are composed of brain-inspired spiking neurons that imitate biological neurons to transmit spikes

*Corresponding author.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

between each other. This allows event-based computation and enables efficient computation on neuromorphic hardware with low energy consumption [14, 15, 16].

However, the supervised training of SNNs is challenging due to the non-differentiable neuron model with discrete spike-generation procedures. Several kinds of methods are proposed to tackle the problem, and recent progress has empirically obtained successful results. Backpropagation through time (BPTT) with surrogate gradients (SG) is one of the mainstream methods which enables the training of deep SNNs with high performance on large-scale datasets (e.g., ImageNet) with extremely low latency (e.g., 4-6 time steps) [6, 10, 11, 13]. These approaches unfold the iterative expression of spiking neurons, backpropagate the errors through time [17], and use surrogate derivatives to approximate the gradient of the spiking function [3, 4, 18, 19, 20, 21, 22, 23]. As a result, during training, they suffer from significant memory costs proportional to the number of time steps, and the optimization with approximated surrogate gradients is not well guaranteed theoretically. Another branch of works builds the closed-form formulation for the spike representation of neurons, e.g. the (weighted) firing rate or spiking time, which is similar to conventional artificial neural networks (ANNs). Then SNNs can be either optimized by calculating the gradients from the equivalent mappings between spike representations [2, 24, 25, 26, 9, 27], or converted from a trained equivalent ANN counterpart [28, 29, 30, 31, 32, 7, 33, 8, 34]. The optimization of these methods is clearer than surrogate gradients. However, they require a larger number of time steps compared to BPTT with SG. Therefore, they suffer from high latency, and more energy consumption is required if the representation is rate-based. Another critical point for both methods is that they are indeed inconsistent with biological online learning, which is also the learning rule on neuromorphic hardware [15].

In this work, we develop a novel approach for training SNNs to achieve high performance with low latency, and maintain the online learning property to pave a path for learning on neuromorphic chips. We call our method online training through time (OTTT). We first derive OTTT from the commonly used BPTT with SG method by analyzing the temporal dependency and proposing to track the presynaptic activities in order to decouple this dependency. With the instantaneous loss calculation, OTTT can perform forward-in-time learning, i.e. calculations are done online in time without computing backward through the time. Then we theoretically analyze the gradients of OTTT and gradients of spike representation-based methods. We show that they have similar expressions and prove that they can provide the similar descent direction for the optimization problem formulated by spike representation. For the feedforward network condition, gradients are easily calculated and analyzed. For the recurrent network condition, we follow the framework in [12] that weighted firing rates will converge to an equilibrium state and gradients can be calculated by implicit differentiation. With this formulation, the gradients correspond to an approximation of gradients calculated by implicit differentiation, which can be proved to be a descent direction for the optimization problem as well [35, 36]. In this way, a connection between OTTT and spike representation-based methods is bridged. Finally, we show that OTTT is in the form of three-factor Hebbian learning rule [37], which could pave a path for online learning on neuromorphic chips. Our contributions include:

1. We propose online training through time (OTTT) for SNNs, which enables forward-in-time learning and only requires constant training memory agnostic to time steps, avoiding the large training memory costs of backpropagation through time (BPTT).
2. We theoretically analyze and connect the gradients of OTTT and gradients based on spike representations, and prove the descent guarantee for optimization under both feedforward and recurrent conditions.
3. We show that OTTT is in the form of three-factor Hebbian learning rule, which could pave a path for on-chip online learning. With OTTT, it is the first time that a connection between BPTT with SG, spike representation-based methods, and biological three-factor Hebbian learning is bridged.
4. We conduct extensive experiments on CIFAR-10, CIFAR-100, ImageNet, and CIFAR10-DVS, which demonstrate the superior results of our methods on large-scale static and neuromorphic datasets in a small number of time steps.

2 Related Work

SNN Training Methods. As for supervised training of SNNs, there are two main research directions. One direction is to build a connection between spike representations (e.g. firing rates) of SNNs with equivalent ANN-like closed-form mappings. With the connection, SNNs can be converted from

ANNs [28, 29, 30, 31, 32, 7, 33, 8, 34, 38], or SNNs can be optimized by gradients calculated from equivalent mappings [2, 24, 25, 26, 9, 27]. Variants following this direction also include [12] which connects feedback SNNs with equilibrium states following fixed-point equations instead of closed-form mappings. These methods have a clearer descent direction for the optimization problem, but require a relatively large number of time steps, suffering from high latency and usually more energy consumption with rate based representation. Another direction is to directly calculate gradients based on the SNN computation. They follow the BPTT framework, and deal with the non-differentiable problem of spiking functions by applying surrogate gradients [3, 4, 18, 19, 20, 21, 6, 23, 10, 11, 13], or computing gradients with respect to spiking times based on the linear assumption [39, 40], or combining both [22]. BPTT with SG can achieve extremely low latency. However, it requires large training memory to maintain the computational graph unfolded along time, and it remains unknown why surrogate gradients work well. [10] empirically observed that surrogate gradients have a high similarity with numerical gradients, but it remains unclear theoretically. And gradients based on spiking times suffer from the "dead neuron" problem [3], so they should be combined with SG in practice [40, 22]. Meanwhile, methods in both directions are inconsistent with biological online learning, i.e. forward-in-time learning, to pave a path for learning on neuromorphic hardware. Differently, our proposed method avoids the above problems and maintain the online property.

Online Training of Neural Networks. In the research of recurrent neural networks (RNNs), there are several alternatives for BPTT to enable online learning. Particularly, real time recurrent learning (RTRL) [41] proposes to propagate partial derivatives of hidden states over parameters through time to enable forward-in-time calculation of gradients. Several recent works improve the memory costs of RTRL with approximation for more practical usage [42, 43, 44]. Another work proposes to online update parameters based on decoupled gradients with regularization at each time step [45]. However, these are all for RNNs and not tailored to SNNs. Several online training methods are proposed for SNNs [46, 47, 48], which are derived in the spirit of RTRL and simplified for SNNs. [49] leverages local losses and ignores temporal dependencies for online local training of SNNs, and [50] directly apply the method in [45] to train SNNs. However, these methods also leverage surrogate gradients without providing theoretical explanation for optimization. Meanwhile, [46, 49] use feedback alignment [51], [47] is limited to single-layer recurrent SNNs, and [48] requires much larger memory costs for eligibility traces, so they cannot scale to large-scale tasks. [50] requires a specially designed neuron model and more computation for parameter regularization, and also does not consider large tasks. Differently, our work explain the descent direction under both feedforward and recurrent conditions with convergent inputs, and is efficient and scalable to large-scale tasks including ImageNet classification.

3 Preliminaries

3.1 Spiking Neural Networks

Spiking neurons are brain-inspired models that transmit information by spike trains. Each neuron maintains a membrane potential u and integrates input spike trains, which will generate a spike once u exceeds a threshold. We consider the commonly used leaky integrate and fire (LIF) model, which describes the dynamics of the membrane potential as:

$$\tau_m \frac{du}{dt} = -(u - u_{rest}) + R \cdot I(t), \quad u < V_{th}, \tag{1}$$

where I is the input current, V_{th} is the threshold, and R and τ_m are resistance and time constant, respectively. A spike is generated when u reaches V_{th} at time t^f, and u is reset to the resting potential $u = u_{rest}$, which is usually set to be zero. The output spike train is defined using the Dirac delta function: $s(t) = \sum_{t^f} \delta(t - t^f)$.

A spiking neural network is composed of connected spiking neurons with connection coefficients. We consider a simple current model $I_i(t) = \sum_j w_{ij} s_j(t) + b_i$, where the subscript i represents the i-th neuron, w_{ij} is the weight from neuron j to neuron i, and b_i is a bias. The discrete computational form is:

$$\begin{cases} u_i\left[t+1\right] = \lambda(u_i[t] - V_{th} s_i[t]) + \sum_j w_{ij} s_j[t] + b_i, \\ \\ s_i[t+1] = H(u_i\left[t+1\right] - V_{th}), \end{cases} \tag{2}$$

where $H(x)$ is the Heaviside step function, $s_i[t]$ is the spike train of neuron i at discrete time step t, and $\lambda < 1$ is a leaky term (typically taken as $1 - \frac{1}{\tau_m}$). The constant R, τ_m, and time step size are absorbed into the weights and bias. The reset operation is implemented by subtraction.

3.2 Previous SNN Training Methods

Spike Representation. The (weighted) firing rate or first spiking time of spiking neurons can be proved to follow ANN-like closed-form transformations [7, 26, 9, 12, 27]. We focus on the weighted firing rate [12, 27] which has connection with OTTT in this work. Define weighted firing rates and weighted average inputs $\mathbf{a}[t] = \frac{\sum_{\tau=1}^{t} \lambda^{t-\tau}\mathbf{s}[\tau]}{\sum_{\tau=1}^{t} \lambda^{t-\tau}}$, $\overline{\mathbf{x}}[t] = \frac{\sum_{\tau=0}^{t} \lambda^{t-\tau}\mathbf{x}[\tau]}{\sum_{\tau=0}^{t} \lambda^{t-\tau}}$ in the discrete condition. Given convergent weighted average inputs $\overline{\mathbf{x}}[t] \to \mathbf{x}^*$, it can be proved that $\mathbf{a}[t] \to \mathbf{a}^* = \sigma\left(\frac{1}{V_{th}}\mathbf{x}^*\right)$ with bounded random error, where σ is a clamp function ($\sigma(x) = \min(\max(0, x), 1)$) in the discrete condition while a ReLU function in the continuous condition. For feedforward networks, the closed-form mapping between successive layers is established based on weighted firing rate after time T: $\mathbf{a}^{l+1}[T] \approx \sigma\left(\frac{1}{V_{th}}\left(\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1}\right)\right)$, and gradients are calculated with such spike representation: $\frac{\partial L}{\partial \mathbf{W}^l} = \frac{\partial L}{\partial \mathbf{a}^N[T]} \prod_{i=N-1}^{l+1} \frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]} \frac{\partial \mathbf{a}^{l+1}[T]}{\partial \mathbf{W}^l}$. For the recurrent condition, $\mathbf{a}[t]$ will converge to an equilibrium state following an implicit fixed-point equation, e.g. $\mathbf{a}^* = \sigma\left(\frac{1}{V_{th}}\left(\mathbf{W}\mathbf{a}^* + \mathbf{F}\mathbf{x}^* + \mathbf{b}\right)\right)$ for a single-layer network with input connections F and contractive recurrent connections W, and gradients can be calculated based on implicit differentiation [12]. Let $\mathbf{a} = f_{\boldsymbol{\theta}}(\mathbf{a})$ denote the fixed-point equation ($\boldsymbol{\theta}$ are parameters). We have $\frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial L}{\partial \mathbf{a}[T]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1} \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \boldsymbol{\theta}}$, where $J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]} = \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{a}[T]}$ is the Jacobian of $f_{\boldsymbol{\theta}}$ at $\mathbf{a}[T]$.

BPTT with SG. BPTT unfolds the iterative update equation in Eq.(2) and backpropagates along the computational graph as shown in Fig. 1(a), 1(c). The gradients with T time steps are calculated by [2]:

$$\frac{\partial L}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{s}^{l+1}[t]} \frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]} \left(\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{W}^l} + \sum_{\tau<t} \prod_{i=t-1}^{\tau} \left(\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{u}^{l+1}[i]} + \frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{s}^{l+1}[i]} \frac{\partial \mathbf{s}^{l+1}[i]}{\partial \mathbf{u}^{l+1}[i]} \right) \frac{\partial \mathbf{u}^{l+1}[\tau]}{\partial \mathbf{W}^l} \right), \tag{3}$$

where $\mathbf{W}^l$ is the weight from layer l to $l+1$ and L is the loss. The non-differentiable terms $\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}$ will be replaced by surrogate derivatives, e.g. derivatives of rectangular or sigmoid functions [4]: $\frac{\partial s}{\partial u} = \frac{1}{a_1}\text{sign}\left(|u - V_{th}| < \frac{a_1}{2}\right)$ or $\frac{\partial s}{\partial u} = \frac{1}{a_2}\frac{e^{(V_{th}-u)/a_2}}{(1+e^{(V_{th}-u)/a_2})^2}$, where a_1 and a_2 are hyperparameters.

4 Online Training Through Time for SNNs

This section contains four sub-sections. In Section 4.1, we introduce our proposed OTTT by decoupling the temporal dependency of BPTT. Then in Section 4.2, we further connect the gradients of OTTT and spike representation-based methods, and prove that OTTT can provide a descent direction for optimization, which is not guaranteed by BPTT with SG. In Section 4.3, we discuss the relationship between OTTT and the three-factor Hebbian learning rule. Implementation details are presented in Section 4.4.

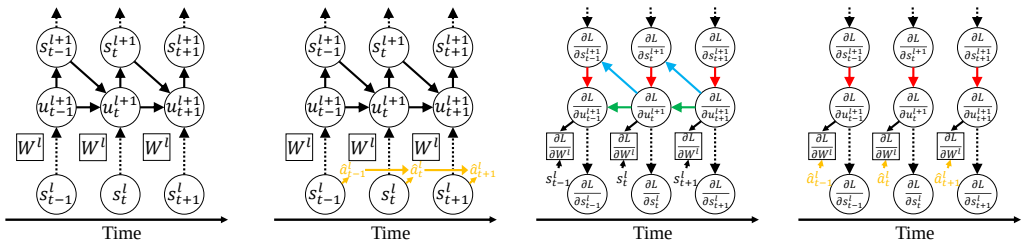
4.1 Derivation of Online Training Through Time

Decouple temporal dependency. As shown in Fig. 1(c), BPTT has to maintain the computational graph of previous time to backpropagate through time. We will decouple such temporal dependency to enable online gradient calculation, as illustrated in Fig. 1(d).

We first focus on the feedforward network condition. In this setting, all temporal dependencies lie in the dynamics of each spiking neuron, i.e. $\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{u}^{l+1}[i]}$ and $\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{s}^{l+1}[i]} \frac{\partial \mathbf{s}^{l+1}[i]}{\partial \mathbf{u}^{l+1}[i]}$ in Eq.(3). We consider the case that we do not apply surrogate derivatives to $\frac{\partial \mathbf{s}^{l+1}[i]}{\partial \mathbf{u}^{l+1}[i]}$ in such temporal dependency. Since the derivative of the Heaviside step function is 0 almost everywhere, we have $\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{s}^{l+1}[i]} \frac{\partial \mathbf{s}^{l+1}[i]}{\partial \mathbf{u}^{l+1}[i]} \approx 0$[3].

[2] Note that we follow the numerator layout convention for derivatives, i.e. $\nabla_{\boldsymbol{\theta}} L = \left(\frac{\partial L}{\partial \boldsymbol{\theta}}\right)^\top$ is the gradient with the same dimension of $\boldsymbol{\theta}$.

[3] Note that this is consistent with some released implementations of BPTT with SG methods which detach the neuron reset operation from the computational graph and do not backpropagate gradients in this path [23, 11].



(a) BPTT Forward (b) OTTT Forward (c) BPTT Backward (d) OTTT Backward

Figure 1: Illustration of the forward and backward procedures of BPTT and OTTT.

Then the dependency only includes $\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{u}^{l+1}[i]}$, which equals $\lambda \mathbf{I}$. Therefore, we have[2]:

$$\frac{\partial L}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{s}^{l+1}[t]} \frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]} \left(\sum_{\tau \leq t} \lambda^{t-\tau} \frac{\partial \mathbf{u}^{l+1}[\tau]}{\partial \mathbf{W}^l} \right), \nabla_{\mathbf{W}^l} L = \sum_{t=1}^{T} \mathbf{g}_{\mathbf{u}^{l+1}}[t] \left(\sum_{\tau \leq t} \lambda^{t-\tau} \mathbf{s}^l[\tau] \right)^\top, \tag{4}$$

where $\mathbf{g}_{\mathbf{u}^{l+1}}[t] = \left(\frac{\partial L}{\partial \mathbf{s}^{l+1}[t]} \frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]} \right)^\top$ is the gradient for $\mathbf{u}^{l+1}[t]$. Based on Eq.(4), we can track presynaptic activities $\hat{\mathbf{a}}^l[t] = \sum_{\tau \leq t} \lambda^{t-\tau} \mathbf{s}^l[\tau]$ for each neuron during the forward procedure by $\hat{\mathbf{a}}^l[t+1] = \lambda \hat{\mathbf{a}}^l[t] + \mathbf{s}^l[t+1]$, so that when given $\mathbf{g}_{\mathbf{u}^{l+1}}[t]$, gradients at each time step can be calculated independently by $\nabla_{\mathbf{W}^l} L[t] = \mathbf{g}_{\mathbf{u}^{l+1}}[t] \hat{\mathbf{a}}^l[t]^\top$ without backpropagation through $\frac{\partial \mathbf{u}^{l+1}[i+1]}{\partial \mathbf{u}^{l+1}[i]}$.

As for the recurrent network condition, there are additional temporal dependencies due to the feedback connections between neurons. If there is feedback connection from layer l_2 to l_1 ($l_2 \geq l_1$), there would be terms such as $\frac{\partial \mathbf{u}^{l_1}[i+1]}{\partial \mathbf{s}^{l_2}[i]} \frac{\partial \mathbf{s}^{l_2}[i]}{\partial \mathbf{u}^{l_2}[i]} \frac{\partial \mathbf{u}^{l_2}[i]}{\partial \mathbf{u}^{l_1}[i]}$ in the calculation of gradients (note that Eq. (3) omit feedback connections for simplicity). We also consider not applying surrogate derivatives to $\frac{\partial \mathbf{s}^{l_2}[i]}{\partial \mathbf{u}^{l_2}[i]}$ in the temporal dependency so that gradients are not calculated in this path. Similar to the feedforward condition, we can derive that the gradients of the general weight $\mathbf{W}^{l_i \to l_j}$ from any layer l_i to any layer l_j can be calculated by $\nabla_{\mathbf{W}^{l_i \to l_j}} L[t] = \mathbf{g}_{\mathbf{u}^{l_j}}[t] \hat{\mathbf{a}}^{l_i}[t]^\top$ at each time step. A theoretical explanation for optimization will be presented in Section 4.2.

Instantaneous Loss and Gradient. Calculating online gradients, e.g. the above $\mathbf{g}_{\mathbf{u}^{l+1}}[t]$ for $\nabla_{\mathbf{W}^l} L[t]$, requires instantaneous computation of the loss at each time step. Previous typical loss for SNNs is based on the firing rate, e.g. $L_{fr} = \mathcal{L}\left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{s}^N[t], \mathbf{y}\right)$, where $\mathbf{y}$ is the label, $\mathbf{s}^N[t]$ is the spike at the last layer, and $\mathcal{L}$ can take cross-entropy loss. This loss depends on all time steps and does not support online gradients. We leverage the instantaneous loss and calculate the above $\mathbf{g}_{\mathbf{u}^{l+1}}[t]$ as:

$$L[t] = \frac{1}{T}\mathcal{L}\left(\mathbf{s}^N[t], \mathbf{y}\right), \quad \mathbf{g}_{\mathbf{u}^{l+1}}[t] = \left(\frac{\partial L[t]}{\partial \mathbf{s}^N[t]} \prod_{i=N-1}^{l+1} \frac{\partial \mathbf{s}^{i+1}[t]}{\partial \mathbf{s}^i[t]} \frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]} \right)^\top. \tag{5}$$

The total loss $L := \sum_{t=1}^{T} L[t]$ is the upper bound of L_{fr} when $\mathcal{L}$ is a convex function such as cross-entropy. Then gradients can be calculated independently at each time step, as shown in Fig. 1(d). We apply surrogate derivatives for $\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}$ in this calculation, which will be explained in Section 4.2.

Since gradients are calculated instantaneously at each time step, OTTT does not require maintaining the unfolded computational graph and only requires constant training memory costs agnostic to time steps. Note that instantaneous gradients of OTTT will be different from BPTT with the instantaneous loss for multi-layer or recurrent networks, as we do not consider future influence in the instantaneous calculation: BPTT considers terms such as $\frac{\partial L[t']}{\partial \mathbf{u}^N[t']} \frac{\partial \mathbf{u}^N[t']}{\partial \mathbf{u}^N[t]} \prod_{i=N-1}^{l} \frac{\partial \mathbf{u}^{i+1}[t]}{\partial \mathbf{u}^i[t]}$ ($t' > t$) for $\mathbf{u}^l[t]$ while we do not. The equivalence of OTTT and BPTT only holds for the last layer, and we do not seek the exact equivalence to BPTT with SG which is theoretically unclear, but will build the connection with spike representations and prove the descent guarantee. Also, note that the tracked presynaptic activities are similar to the biologically plausible "eligibility traces" in the literature [46, 47, 52], and we will provide a more solid theoretical grounding for optimization in Section 4.2.

4.2 Connection with Spike Representation-Based Methods for Descent Directions

In this section, we connect gradients of OTTT and spike representation-based methods, and prove that OTTT can provide a descent direction for optimization under feedforward and recurrent conditions with convergent inputs.

Feedforward Networks. As introduced in Section 3.2, with convergent inputs, methods based on spike representations establish closed-form mappings between successive layers with weighted firing rate $\mathbf{a}[t] = \frac{\sum_{\tau=1}^{t} \lambda^{t-\tau}\mathbf{s}[\tau]}{\sum_{\tau=1}^{t} \lambda^{t-\tau}}$ as $\mathbf{a}^{l+1}[T] \approx \sigma\left(\frac{1}{V_{th}}\left(\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1}\right)\right)$, and calculate gradients by $\frac{\partial L}{\partial \mathbf{W}^l} = \frac{\partial L}{\partial \mathbf{a}^N[T]} \prod_{i=N-1}^{l+1} \frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]} \frac{\partial \mathbf{a}^{l+1}[T]}{\partial \mathbf{W}^l}$. Note that $\mathbf{a}[t]$ is similar to the tracked presynaptic activities $\hat{\mathbf{a}}[t] = \sum_{\tau=1}^{t} \lambda^{t-\tau}\mathbf{s}[\tau]$ in OTTT. We can obtain:

$$(\nabla_{\mathbf{W}^l} L_{sr})_{sr} = \sum_{t=1}^{T} \left(\left(\frac{1}{T} \frac{1}{\lambda^{T-t}} \frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]} \prod_{i=N-1}^{l+1} \frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]} \right)^\top \odot \mathbf{d}^{l+1}[T] \right) \hat{\mathbf{a}}^l[T]^\top, \tag{6}$$

where L_{sr} is the loss based on spike representation, $\mathbf{d}^{l+1}[T] = \sigma'\left(\frac{1}{V_{th}}\left(\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1}\right)\right)$, and '$\odot$' is element-wise product. The detailed derivation can be found in Appendix A.

It can be easily seen that Eq. (6) has a similar form as gradients in Eq. (4), (5), and we build the connection between them in three steps. For sake of clarify, in the following, we denote gradients of OTTT and spike representation by $\nabla_{\mathbf{W}^l} L$ and $(\nabla_{\mathbf{W}^l} L_{sr})_{sr}$, respectively.

In the first step, we can use appropriate surrogate derivatives of $\frac{\partial \mathbf{s}_i^{l+1}[t]}{\partial \mathbf{u}_i^{l+1}[t]} (t = 1, \cdots, T)$ to approximate $\mathbf{d}_i^{l+1}[T]$. As introduced in Section 3.2, σ is a clamp function ($\sigma(x) = \min(\max(0, x), 1)$) in the discrete condition while a ReLU function in the continuous condition. Then $\mathbf{d}^{l+1}[T]$ almost equals sign $\left(|\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1} - V_{th}| < V_{th}\right)$ if we slightly relax the clamp bound caused by the discretization, and this can be approximated by diag $\left(\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right) = \text{sign}\left(|\mathbf{u}^{l+1}[t] - V_{th}| < V_{th}\right)$ at each time step. Note that surrogate derivatives here approximate the well-defined derivative of the mapping function between $\mathbf{a}[T]$, rather than the pseudo derivative of the non-differentiable Heaviside step function as in BPTT with SG. The approximation is exact except the rare case that averagely a neuron generate spikes (i.e. the average input is positive) while sometimes the membrane potential is less than u_{rest} (or the reverse). For simplicity in the following theoritical analysis, we assume the equivalence between surrogate derivatives and $\mathbf{d}[T]$.

Assumption 1. $\forall l = 1, \cdots, N, t = 1, \cdots, T, \text{diag}\left(\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right) = \mathbf{d}^{l+1}[\mathrm{T}]$.

In the second step, we take L_{sr} in Eq. (6) as $L_{srup} = \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \sum_{t=1}^{T} \lambda^{T-t} \mathcal{L}(\mathbf{s}^N[t], \mathbf{y})$ so that $\frac{1}{T} \frac{1}{\lambda^{T-t}} \frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]}$ in Eq. (6) aligns with $\frac{\partial L[t]}{\partial \mathbf{s}^N[t]}$ in Eq. (5) except a constant term. Note that L_{srup} is an upper bound of the common loss $L'_{sr} = \mathcal{L}(\mathbf{a}^N[T], \mathbf{y}) = \mathcal{L}(\frac{\sum_{t=1}^{T} \lambda^{T-t}\mathbf{s}^N[t]}{\sum_{t=1}^{t} \lambda^{T-t}}, \mathbf{y})$ if $\mathcal{L}$ is a convex function. Then let $\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] = \left(\frac{\partial \mathcal{L}(\mathbf{s}^N[t], \mathbf{y})}{\partial \mathbf{s}^N[t]} \prod_{i=N-1}^{l+1} \frac{\partial \mathbf{s}^{i+1}[t]}{\partial \mathbf{s}^i[t]} \frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right)^\top$, with Assumption 1 we have: $\nabla_{\mathbf{W}^l} L = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \hat{\mathbf{a}}^l[t]^\top$ and $(\nabla_{\mathbf{W}^l} L_{sr})_{sr} = \frac{1}{T} \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \hat{\mathbf{a}}^l[T]^\top$. Please refer to Appendix A for details.

In the third step, we handle the remaining difference that $\nabla_{\mathbf{W}^l} L$ leverages instantaneous presynaptic activities $\hat{\mathbf{a}}[t]$ during calculation while $(\nabla_{\mathbf{W}^l} L_{sr})_{sr}$ uses the final $\hat{\mathbf{a}}^l[T]$ after time T. Note that the weighted firing rate gradually converges $\mathbf{a}[t] \to \mathbf{a}^*$ with bounded random error. Suppose the errors $\boldsymbol{\epsilon}^l[t] = \mathbf{a}^l[t] - \mathbf{a}^l[T]$ are small ($l = 0$ represents inputs, i.e. $\mathbf{a}^0[t] = \overline{\mathbf{x}}[t]$), then we have that $-\nabla_{\mathbf{W}^l} L$ can provide a descent direction, as shown in Theorem 1.

Theorem 1. *If Assumption 1 holds, $V_{th} = 1$, and the errors $\boldsymbol{\epsilon}^l[t] = \mathbf{a}^l[t] - \mathbf{a}^l[T]$ are small such that $\left\|\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \boldsymbol{\epsilon}^l[t]^\top\right\| < \left\|\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \mathbf{a}^l[T]^\top\right\| - \left\|\sum_{t=1}^{T} \frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \mathbf{a}^l[t]^\top\right\|$ when $(\nabla_{\mathbf{W}^l} L_{sr})_{sr} \neq \mathbf{0}$, then we have $\langle \nabla_{\mathbf{W}^l} L, (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \rangle > 0$.*

For the proof and discussion of the assumption please refer to Appendix A. With this conclusion, we can explain the descent direction of gradient descent by OTTT for the optimization problem

formulated by spike representation. Some random error can be viewed as randomness for stochastic optimization.

Recurrent Networks. For networks with feedback connections, we first consider the single-layer condition for simplicity (see Appendix A for general conditions). We consider feedforward connections $\mathbf{F}$ from inputs to neurons and contractive recurrent connections $\mathbf{W}$ between neurons. As introduced in Section 3.2, given convergent inputs $\overline{\mathbf{x}}[t] \to \mathbf{x}^*$, $\mathbf{a}[t]$ of neurons will converge to an equilibrium state $\mathbf{a}^* = f_{\boldsymbol{\theta}}(\mathbf{a}^*) = \sigma\left(\frac{1}{V_{th}}\left(\mathbf{W}\mathbf{a}^* + \mathbf{F}\mathbf{x}^* + \mathbf{b}\right)\right)$ with bounded random error, and gradients are calculated as $(\nabla_{\boldsymbol{\theta}} L_{sr})_{sr} = \left(\frac{\partial L_{sr}}{\partial \boldsymbol{\theta}}\right)^\top = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \boldsymbol{\theta}}\right)^\top$, where $\boldsymbol{\theta} \in \{\mathbf{W}, \mathbf{F}, \mathbf{b}\}$. We consider replacing the inverse Jacobian by an identity matrix: $(\widetilde{\nabla_{\boldsymbol{\theta}} L_{sr}})_{sr} = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \boldsymbol{\theta}}\right)^\top$. Previous works have proved that this gradient can provide a descent direction for the optimization problem [35, 36], i.e. $\left\langle (\widetilde{\nabla_{\boldsymbol{\theta}} L_{sr}})_{sr}, (\nabla_{\boldsymbol{\theta}} L_{sr})_{sr}\right\rangle > 0$. It has a similar form as the OTTT gradient: $\nabla_{\mathbf{W}} L = \sum_{t=1}^{T} \mathbf{g}_{\mathbf{u}}[t]\hat{\mathbf{a}}[t]^\top$ and $(\widetilde{\nabla_{\mathbf{W}} L_{sr}})_{sr} = \sum_{t=1}^{T}\left(\frac{1}{T}\frac{1}{\lambda^{T-t}}\frac{\partial L_{sr}}{\partial \mathbf{s}[t]}^\top \odot \mathbf{d}[T]\right)\hat{\mathbf{a}}[T]^\top$. Similarly, we can prove the descent guarantee for OTTT as shown in Theorem 2 . For details refer to Appendix A.

Theorem 2. *If Assumption 1 holds, $V_{th} = 1$, $\left\|J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right\| \leq \eta < \frac{\sigma_{min}^2}{\sigma_{max}^2}$, where σ_{max} and σ_{min} are the maximal and minimal singular value of $\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}|_{\mathbf{a}[T]}$, and the errors $\boldsymbol{\epsilon}^1[t] = \mathbf{a}[t] - \mathbf{a}[T], \boldsymbol{\epsilon}^0[t] = \overline{\mathbf{x}}[t] - \overline{\mathbf{x}}[T]$ are small such that $\left\|\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}}[t]\boldsymbol{\epsilon}^l[t]^\top\right\| < \frac{\sigma_{min}^2 - \eta\sigma_{max}^2}{\sigma_{max}}\left\|\sum_{t=1}^{T}\frac{\partial \mathcal{L}(\mathbf{s}[t],\mathbf{y})}{\partial \mathbf{s}[t]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\right\| - \left\|\sum_{t=1}^{T}\frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T}\hat{\mathbf{g}}_{\mathbf{u}}[t]\mathbf{a}^l[t]^\top\right\|$ (where $l = 0, 1$, $\mathbf{a}^1[t]$ and $\mathbf{a}^0[t]$ represent $\mathbf{a}[t]$ and $\overline{\mathbf{x}}[t]$, respectively) when $(\nabla_{\boldsymbol{\theta}} L_{sr})_{sr} \neq \mathbf{0}$, then we have $\langle \nabla_{\boldsymbol{\theta}} L, (\nabla_{\boldsymbol{\theta}} L_{sr})_{sr}\rangle > 0$, where $\boldsymbol{\theta}$ are parameters in the network.*

4.3 Connection with Three-factor Hebbian Learning Rule

By explicitly writing the instantaneous gradients of OTTT for the general weight from layer l_i to l_j, $\nabla_{\mathbf{W}^{l_i \to l_j}} L[t] = \mathbf{g}_{\mathbf{u}^{l_j}}[t]\hat{\mathbf{a}}^{l_i}[t]^\top$, and dive into connections between any two neurons i and j, we have:

$$\nabla_{W_{i,j}} L[t] = \hat{a}_i[t] f(u_j[t])\delta_j[t], \tag{7}$$

where $\hat{a}_i[t]$ is the tracked presynaptic activity, $f(u_j[t])$ is the surrogate derivative function which can represent the change rate of the postsynaptic activity as analyzed in Section 4.2, and $\delta_j[t] = g_{u_j}[t]$ is the gradient for $u_j[t]$ which represents a global modulator. This is a kind of three-factor Hebbian learning rule [37] and the weight can be updated locally with a global signal. The error signal $\delta_j[t]$ can be propagated in an error feedback path simultaneously with feedforward propagation, which is shown biologically plausible with high-frequency bursts [53]. Note that the analysis in Section 4.2 still holds if we consider the delay of the propagation of the error signal, i.e. the update is based on $\hat{a}_i[t + \Delta t] f(u_j[t + \Delta t])\delta_j[t]$.

4.4 Implementation Details

As introduced in Section 4.1, we will calculate instantaneous gradients $\nabla_{\mathbf{W}^{l_i \to l_j}} L[t] = \mathbf{g}_{\mathbf{u}^{l_j}}[t]\hat{\mathbf{a}}^{l_i}[t]^\top$ at each time step. We can choose to immediately update parameters before the calculation of the next time step, which we denote as OTTT$_O$, or we can accumulate the gradients by T time steps and then update parameters, which we denote as OTTT$_A$. For OTTT$_O$, we assume that the online update is small and has negligible affects for the following calculation. Pseudo-codes are in Appendix B.

An important issue in practice is that previous BPTT with SG works leverage batch normalization (BN) along the temporal dimension to achieve high performance with extremely low latency on large-scale datasets [6, 10, 11, 13], which requires calculating the mean and variance statistics for all time steps during the forward procedure. This technique intrinsically prevents online gradients and has to suffer from large memory costs. To overcome this shortcoming, we do not use BN, but borrow the idea from normalization-free ResNets (NF-ResNets) [54, 55] to replace batch normalization by scaled weight standardization (sWS) [56]. sWS standardizes weights by $\hat{\mathbf{W}}_{i,j} = \gamma \cdot \frac{\mathbf{W}_{i,j} - \mu_{\mathbf{W}_{i,\cdot}}}{\sigma_{\mathbf{W}_{i,\cdot}}\sqrt{N}}$, and the scale γ is determined by analyzing the signal propagation with different activation functions. We apply sWS for VGG [57] and NF-ResNet architectures in our experiments. For details please refer to Appendix C.

5 Experiments

In this section, we conduct extensive experiments on CIFAR-10 [58], CIFAR100 [58], ImageNet [59], CIFAR10-DVS [60], and DVS128-Gesture [61] to demonstrate the superior performance of our proposed method on large-scale static and neuromorphic datasets. We leverage the VGG network architecture (64C3-128C3-AP2-256C3-256C3-AP2-512C3-512C3-AP2-512C3-512C3-GAP-FC) for experiments on CIFAR-10, CIFAR-100, CIFAR10-DVS, and DVS128-Gesture, and the NF-ResNet-34 [54] network architecture for experiments on ImageNet. For all our SNN models, we set $V_{th} = 1$ and $\lambda = 0.5$. Please refer to Appendix C for training details.

5.1 Comparison of Training Memory Costs

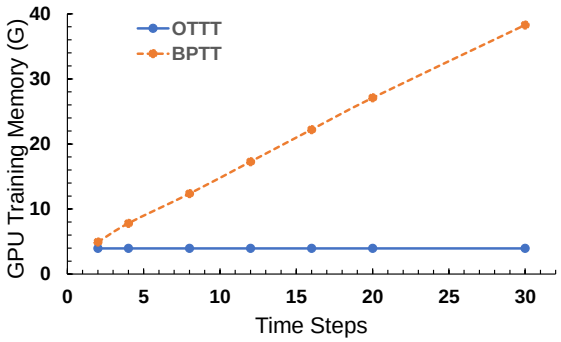


Figure 2: Comparison of training memory costs between OTTT and BPTT under different time steps.

A major advantage of OTTT over BPTT is that OTTT does not require backpropagation along the temporal dimension and therefore only requires constant training memory costs agnostic to time steps, which avoids the large memory costs of BPTT. We verify this by training the VGG network on CIFAR-10 with batch size 128 under different time steps and calculating the memory costs on the GPU. As shown in Fig. 2, the training memory of BPTT grows linearly with time steps, while OTTT maintains the constant memory (both OTTT_A and OTTT_O). Even with a small number of time steps, e.g. 6, OTTT can reduce the memory costs by $2 \sim 3\times$. This advantage may also allow training acceleration of SNNs by larger batch sizes with the same computational resources.

5.2 Comparison of Performance

We conduct experiments on both large-scale static and neuromorphic datasets. We first verify the effectiveness of the surrogate derivative $\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]} = \text{sign}\left(|\mathbf{u}^{l+1}[t] - V_{th}| < V_{th}\right)$ in Section 4.2. For the VGG (sWS) network on CIFAR-10, OTTT_A and OTTT_O achieves 91.46% and 92.47% test accuracy respectively. We empirically observe that applying the sigmoid-like surrogate derivative achieves a higher generalization performance, e.g. OTTT_A and OTTT_O achieves 93.58% and 93.73% test accuracy respectively under the same random seed, and a possible reason may be that this introduces some noise for the approximation to regularize the training and improve the generalization. Therefore, in the following performance evaluation, we take sigmoid-like surrogate derivative for OTTT. As shown in Table 1, both OTTT_A and OTTT_O achieve satisfactory performance on all datasets, and compared with BPTT under the same training settings, OTTT achieves higher performance. The proposed OTTT also achieves promising performance on all datasets compared with other representative conversion and direct training methods. Besides, it shows that the performance gap between our SNN model and ANN is around 0.7% and 2.08% on CIFAR-10 and CIFAR-100, respectively. Usually, SNNs with a very small number of time steps do not reach the performance of equivalent ANNs due to the information propagation with discrete spikes rather than floating-point numbers. The results of our model with 6 time steps are competitive.

We also evaluate our method on the DVS128-Gesture dataset, which is more time-varying with different hand gestures recorded by a DVS camera. As shown in Table 2, our method can achieve the same high performance as BPTT. While our theoretical analysis mainly focus on convergent inputs (e.g. static images or neuromorphic inputs converted from images like CIFAR10-DVS), the results show that our method can also work well for time-varying inputs.

Table 1: Performance on CIFAR-10, CIFAR-100, ImageNet, and CIFAR10-DVS. Results are based on 3 runs of experiments (except ImageNet). Our OTTT is mainly compared with BPTT under the same settings, and is also compared with other representative conversion and direct training methods.

Dataset	Method	Network structure	Params	Time steps	Mean±Std (Best)
CIFAR-10	ANN-SNN [7]	VGG-16	40M	16	(92.29%)
	BPTT [6]	ResNet-19 (tdBN)	14.5M	6	(93.16%)
	BPTT [23]	9-layer CNN (PLIF, BN)	36M	8	(93.50%)
	BPTT	VGG (sWS)	9.2M	6	92.78±0.34% (93.23%)
	OTTT$_A$ (ours)	VGG (sWS)	9.2M	6	**93.52±0.06% (93.58%)**
	OTTT$_O$ (ours)	VGG (sWS)	9.2M	6	**93.49±0.17% (93.73%)**
	ANN	VGG (sWS)	9.2M	N.A.	(94.43%)
CIFAR-100	ANN-SNN [7]	VGG-16	40M	400-600	(70.55%)
	Hybrid Training [31]	VGG-11	36M	125	(67.87%)
	DIET-SNN [62]	VGG-16	40M	5	(69.67%)
	BPTT	VGG (sWS)	9.3M	6	69.06±0.07% (69.15%)
	OTTT$_A$ (ours)	VGG (sWS)	9.3M	6	**71.05±0.04% (71.11%)**
	OTTT$_O$ (ours)	VGG (sWS)	9.3M	6	**71.05±0.06% (71.11%)**
	ANN	VGG (sWS)	9.3M	N.A.	(73.19%)
ImageNet	ANN-SNN [8]	ResNet-34	22M	32	(64.54%)
	Hybrid Training [31]	ResNet-34	22M	250	(61.48%)
	BPTT [6]	ResNet-34 (tdBN)	22M	6	(63.72%)
	OTTT$_A$ (ours)	NF-ResNet-34	22M	6	**(65.15%)**
	OTTT$_O$ (ours)	NF-ResNet-34	22M	6	**(64.16%)**
DVS-CIFAR10	Tandem Learning [9]	CifarNet	45M	20	(65.59%)
	BPTT [6]	ResNet-19 (tdBN)	14.5M	10	(67.80%)
	BPTT [23]	7-layer CNN (PLIF, BN)	1.1M	20	(74.80%)
	BPTT	VGG (sWS)	9.2M	10	72.60±1.26% (73.90%)
	OTTT$_A$ (ours)	VGG (sWS)	9.2M	10	**76.27±0.05% (76.30%)**
	OTTT$_O$ (ours)	VGG (sWS)	9.2M	10	**76.63±0.34% (77.10%)**

Table 2: Performance on DVS128-Gesture.

Method	Network structure	Time steps	Accuracy
SLAYER [3]	8-layer CNN	300	93.64±0.49%
DECOLLE [49]	3-layer CNN	1800	95.54±0.16%
BPTT [23]	8-layer CNN (PLIF, BN)	20	97.57%
BPTT [23]	8-layer CNN (LIF, BN)	20	96.88%
BPTT	VGG (sWS)	20	96.88%
OTTT$_A$ (ours)	VGG (sWS)	20	96.88%

5.3 Effectiveness for Recurrence

As introduced in Section 4, the proposed OTTT is also valid for networks with feedback connections. Previous works have shown that adding feedback connections can improve the performance of SNNs without much additional costs, especially on the CIFAR-100 datasets [12, 63]. Therefore, we conduct experiments on CIFAR-100 with the VGG-F network architecture which simply adds a feedback connection from the last feature layer to the first feature layer following [12], and this weight is zero-intialized. As shown in Table 3, the training of VGG-F is valid and VGG-F achieves a higher performance than VGG due to the introduction of feedback connections. Results in Appendix D show that the improvement of OTTT from feedback connections is more significant than that of BPTT. The architectures with feedback connections can be further improved with neural architecture search [63].

5.4 Effectiveness for Training with Batch Size 1

To further study the online training, i.e. not only online in time but also one sample per training, which is consistent with biological learning and learning on neuromorphic hardware, we verify the effectiveness for training with batch size 1. The VGG network on CIFAR-10 is studied, and batch size 1 is compared with the default batch size 128 under the same random seed. Models are only trained for 20 epochs due to the relatively long training time with batch size 1. As shown in Table 4, training with one sample per iteration is still valid, indicating the potential to conduct full online training with the proposed OTTT.

Table 3: Performance on CIFAR-100 for VGG and VGG-F trained by OTTT_O. Results are based on 3 runs of experiments.

Network structure	Params	Mean±Std (Best)
VGG	9.3M	71.05±0.06% (71.11%)
VGG-F	9.6M	72.63±0.23% (72.94%)

Table 4: Performance of VGG on CIFAR-10 with different batch sizes for 20 epochs under the same random seed.

Method	Batch Size	Accuracy
OTTT_A / OTTT_O	128	88.20% / 88.62%
OTTT_A / OTTT_O	1	88.07% / 88.50%

5.5 Influence of Inference Time Steps

We study the influence of inference time steps on ImageNet as shown in Fig. 3. It illustrates that the model trained with time step 6 can achieve higher performance with more inference time steps.

5.6 Firing Rate Statistics

We study the firing rate statistics of the models trained by OTTT and BPTT, as shown in Fig. 4. It demonstrates that models trained by OTTT have higher firing rates in first layers while lower firing rates in later layers compared with BPTT. Overall the firing rate is around 0.19 and with 6 time steps each neuron averagely generate 1.1 spikes, indicating the low energy consumption. Considering that each neuron has more synaptic operations in later layers than first layers (because the channel size is increasing with layers), the synaptic operations of models trained by OTTT and BPTT are about the same (1.98×10^8 vs 1.93×10^8). More results please refer to Appendix D.

Figure 3: Influence of inference time steps for the model trained with 6 time steps on ImageNet.

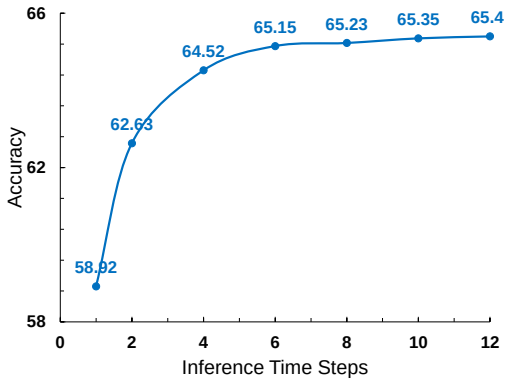
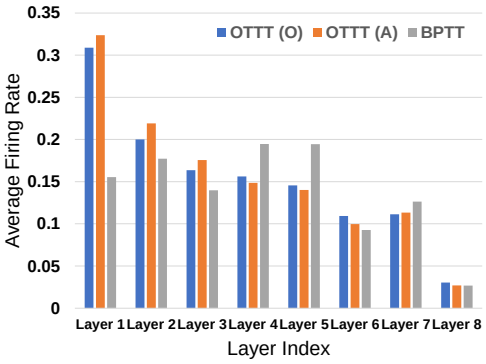


Figure 4: The average firing rates for the models trained by OTTT and BPTT on CIFAR-10.



6 Conclusion

In this work, we propose a new training method, online training through time (OTTT), for spiking neural networks. We first derive OTTT from BPTT with SG by decoupling the temporal dependency with the tracked pre-synaptic activities, which only requires constant training memory agnostic to time steps and avoids the large training memory costs of BPTT. Then we theoretically analyze and connect the gradients of OTTT and gradients of methods based on spike representations, and prove the descent guarantee of OTTT for the optimization problem under both feedforward and recurrent network conditions. Additionally, we show that OTTT is in the form of three-factor Hebbian learning rule, which is the first to connect BPTT with SG, spike representation-based methods, and biological learning rules. Extensive experiments demonstrate the superior performance of our methods on large-scale static and neuromorphic datasets in a small number of time steps.

Acknowledgement

Z. Lin was supported by the major key project of PCL (No. PCL2021A12), the NSF China (No.s 62276004 and 61731018), and Project 2020BD006 supported by PKU-Baidu Fund.

References

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

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

[3] Sumit Bam Shrestha and Garrick Orchard. Slayer: spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, 2018.

[4] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018.

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

[6] 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*, 2021.

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

[8] Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *International Conference on Machine Learning*, 2021.

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

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

[11] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *Advances in Neural Information Processing Systems*, 2021.

[12] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In *Advances in Neural Information Processing Systems*, 2021.

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

[14] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[15] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[16] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid Tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

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

[18] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, 2018.

[19] Yingyezhe Jin, Wenrui Zhang, and Peng Li. Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[20] 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*, 2019.

[21] 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.

[22] 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*, 2020.

[23] 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 (ICCV)*, 2021.

[24] Johannes C Thiele, Olivier Bichler, and Antoine Dupret. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. In *International Conference on Learning Representations*, 2020.

[25] Hao Wu, Yueyi Zhang, Wenming Weng, Yongting Zhang, Zhiwei Xiong, Zheng-Jun Zha, Xiaoyan Sun, and Feng Wu. Training spiking neural networks with accumulated spiking flow. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[26] 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*, 2021.

[27] Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[28] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with LIF neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[29] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017.

[30] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

[31] 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*, 2020.

[32] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *European Conference on Computer Vision*, 2020.

[33] Zhanglu Yan, Jun Zhou, and Weng-Fai Wong. Near lossless transfer learning for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[34] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238, 2021.

[35] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb: Jacobian-free backpropagation for implicit networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[36] Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. In *Advances in Neural Information Processing Systems*, 2021.

[37] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016.

[38] Qingyan Meng, Shen Yan, Mingqing Xiao, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Training much deeper spiking neural networks with a small number of time-steps. *Neural Networks*, 153:254–268, 2022.

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

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

[41] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[42] Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. In *International Conference on Learning Representations*, 2018.

[43] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random kronecker factors. In *Advances in Neural Information Processing Systems*, 2018.

[44] Jacob Menick, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves. Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations*, 2021.

[45] Anil Kag and Venkatesh Saligrama. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, 2021.

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

[47] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications*, 11(1):1–15, 2020.

[48] Thomas Bohnstingl, Stanisław Woźniak, Angeliki Pantazi, and Evangelos Eleftheriou. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[49] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14:424, 2020.

[50] Bojian Yin, Federico Corradi, and Sander M Bohte. Accurate online training of dynamical spiking neural networks through forward propagation through time. *arXiv preprint arXiv:2112.11231*, 2021.

[51] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

[52] James M Murray. Local online learning in recurrent networks with random feedback. *Elife*, 8:e43299, 2019.

[53] Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature Neuroscience*, 24(7):1010–1019, 2021.

[54] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *International Conference on Learning Representations*, 2021.

[55] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, 2021.

[56] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[58] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

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

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

[61] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[62] Nitin Rathi and Kaushik Roy. Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[63] Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, and Priyadarshini Panda. Neural architecture search for spiking neural networks. In *European Conference on Computer Vision*, 2022.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.

[65] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[66] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

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

[68] Wenrui Zhang and Peng Li. Spike-train level backpropagation for training deep recurrent spiking neural networks. In *Advances in Neural Information Processing Systems*, 2019.

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] See Appendix E.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix E.
   (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] See Section 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
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] See Supplementary Materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 5.
   (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] See Appendix C.
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] See Section 5.
   (b) Did you mention the license of the assets? [Yes] See Appendix C.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] We do not use new assets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use the existing common datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use the existing common datasets.
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]

A Detailed Derivation and Proofs

A.1 Derivation of Eq. (6)

Since $\mathbf{a}[t] = \frac{\sum_{\tau=1}^{t} \lambda^{t-\tau}\mathbf{s}[\tau]}{\sum_{\tau=1}^{t} \lambda^{t-\tau}}, \hat{\mathbf{a}}[t] = \sum_{\tau=1}^{t} \lambda^{t-\tau}\mathbf{s}[\tau], \mathbf{a}^{l+1}[T] \approx \sigma\left(\frac{1}{V_{th}}\left(\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1}\right)\right)$, $\mathbf{d}^{l+1}[T] = \sigma'\left(\frac{1}{V_{th}}\left(\mathbf{W}^l\mathbf{a}^l[T] + \mathbf{b}^{l+1}\right)\right)$, $\left(\frac{\partial L_{sr}}{\partial \mathbf{W}^l}\right)_{sr} = \frac{\partial L_{sr}}{\partial \mathbf{a}^N[T]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\frac{\partial \mathbf{a}^{l+1}[T]}{\partial \mathbf{W}^l}$, and we have $\frac{\partial L_{sr}}{\partial \hat{\mathbf{a}}^N[T]} = \frac{1}{\lambda^{T-t}}\frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]}(\forall 1 \le t \le T)$[4], $\frac{\partial L_{sr}}{\partial \hat{\mathbf{a}}^N[T]} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{\lambda^{T-t}}\frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]}$, we can obtain:

$$
\begin{aligned}
\left(\nabla_{\mathbf{W}^l} L_{sr}\right)_{sr} &= \left(\frac{\partial L_{sr}}{\partial \mathbf{W}^l}\right)_{sr}^{\top} = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}^N[T]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\frac{\partial \mathbf{a}^{l+1}[T]}{\partial \mathbf{W}^l}\right)^{\top} \\
&= \left(\left(\frac{\partial L_{sr}}{\partial \mathbf{a}^N[T]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\right)^{\top} \odot \mathbf{d}^{l+1}[T]\right)\mathbf{a}^l[T]^{\top} \\
&= \left(\left(\frac{\partial L_{sr}}{\partial \hat{\mathbf{a}}^N[T]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\right)^{\top} \odot \mathbf{d}^{l+1}[T]\right)\hat{\mathbf{a}}^l[T]^{\top} \\
&= \sum_{t=1}^{T}\left(\left(\frac{1}{T}\frac{1}{\lambda^{T-t}}\frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\right)^{\top} \odot \mathbf{d}^{l+1}[T]\right)\hat{\mathbf{a}}^l[T]^{\top}.
\end{aligned}
\tag{8}
$$

A.2 Proof of Theorem 1

In this subsection, we prove Theorem 1 with Assumption 1.

Assumption 1. $\forall l = 1, \cdots, N, t = 1, \cdots, T, \mathrm{diag}\left(\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right) = \mathbf{d}^{l+1}[\mathrm{T}]$.

Theorem 1. *If Assumption 1 holds, $V_{th} = 1$, and the errors $\boldsymbol{\epsilon}^l[t] = \mathbf{a}^l[t] - \mathbf{a}^l[T]$ are small such that $\left\|\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\boldsymbol{\epsilon}^l[t]^{\top}\right\| < \left\|\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\mathbf{a}^l[T]^{\top}\right\| - \left\|\sum_{t=1}^{T}\frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\mathbf{a}^l[t]^{\top}\right\|$ when $\left(\nabla_{\mathbf{W}^l} L_{sr}\right)_{sr} \neq \mathbf{0}$, then we have $\left\langle \nabla_{\mathbf{W}^l} L, \left(\nabla_{\mathbf{W}^l} L_{sr}\right)_{sr}\right\rangle > 0$.*

Proof. As described in Sections 4.1 and 4.2, for gradients of OTTT, we have $\nabla_{\mathbf{W}^l} L = \sum_{t=1}^{T}\mathbf{g}_{\mathbf{u}^{l+1}}[t]\hat{\mathbf{a}}^l[t]^{\top}$, $L := \sum_{t=1}^{T} L[t] = \sum_{t=1}^{T}\frac{1}{T}\mathcal{L}\left(\mathbf{s}^N[t], \mathbf{y}\right)$, $\mathbf{g}_{\mathbf{u}^{l+1}}[t] = \left(\frac{\partial L[t]}{\partial \mathbf{s}^N[t]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{s}^{i+1}[t]}{\partial \mathbf{s}^i[t]}\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right)^{\top}$; for gradients based on spike representation, we have $\left(\nabla_{\mathbf{W}^l} L_{sr}\right)_{sr} = \sum_{t=1}^{T}\left(\left(\frac{1}{T}\frac{1}{\lambda^{T-t}}\frac{\partial L_{sr}}{\partial \mathbf{s}^N[t]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{a}^{i+1}[T]}{\partial \mathbf{a}^i[T]}\right)^{\top} \odot \mathbf{d}^{l+1}[T]\right)\hat{\mathbf{a}}^l[T]^{\top}$ and consider $L_{sr} = \frac{1}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\sum_{t=1}^{T}\lambda^{T-t}\mathcal{L}(\mathbf{s}^N[t], \mathbf{y})$. Let $\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] = \left(\frac{\partial \mathcal{L}(\mathbf{s}^N[t], \mathbf{y})}{\partial \mathbf{s}^N[t]}\prod_{i=N-1}^{l+1}\frac{\partial \mathbf{s}^{i+1}[t]}{\partial \mathbf{s}^i[t]}\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{u}^{l+1}[t]}\right)^{\top}$, we have $\nabla_{\mathbf{W}^l} L = \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\hat{\mathbf{a}}^l[t]^{\top}$. With Assumption 1, we have $\frac{\partial \mathbf{s}_i^{l+1}[t]}{\partial \mathbf{u}_i^{l+1}[t]} = \mathbf{d}_i^{l+1}[T]$ and thus $\frac{\partial \mathbf{s}^{l+1}[t]}{\partial \mathbf{s}^l[t]} = \frac{\partial \mathbf{a}^{l+1}[t]}{\partial \mathbf{a}^l[t]}$ (because $\frac{\partial \mathbf{s}_j^{l+1}[t]}{\partial \mathbf{s}_i^l[t]} = \frac{\partial \mathbf{s}_j^{l+1}[t]}{\partial \mathbf{u}_j^{l+1}[t]} \cdot \mathbf{W}_{i,j} = \mathbf{d}_j^{l+1}[T] \cdot \mathbf{W}_{i,j} = \frac{\partial \mathbf{a}_j^{l+1}[t]}{\partial \mathbf{a}_i^l[t]}$). So we can derive that $\left(\nabla_{\mathbf{W}^l} L_{sr}\right)_{sr} = \frac{1}{T}\frac{1}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\hat{\mathbf{a}}^l[T]^{\top} = \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\mathbf{a}^l[T]^{\top}$.

We consider $\widehat{\nabla_{\mathbf{W}^l} L} = \frac{1}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\nabla_{\mathbf{W}^l} L = \frac{1}{T}\frac{1}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\hat{\mathbf{a}}^l[t]^{\top} = \frac{1}{T}\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\frac{\sum_{\tau=0}^{t-1}\lambda^{\tau}}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\mathbf{a}^l[t]^{\top}$. Since the errors $\boldsymbol{\epsilon}^l[t] = \mathbf{a}^l[t] - \mathbf{a}^l[T]$ are small such that $\left\|\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\boldsymbol{\epsilon}^l[t]^{\top}\right\| < \left\|\sum_{t=1}^{T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\mathbf{a}^l[T]^{\top}\right\| - \left\|\sum_{t=1}^{T}\frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T}\hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t]\mathbf{a}^l[t]^{\top}\right\|$ when

[4] Note that we can treat $\mathbf{s}^N[t]$ independent with each other, if we consider taking the derivative of the Heaviside step function as 0 in this calculation and therefore $\frac{\partial \mathbf{s}^N[t+1]}{\partial \mathbf{s}^N[t]} = \frac{\partial \mathbf{s}^N[t+1]}{\partial \mathbf{u}^N[t+1]}\frac{\partial \mathbf{u}^N[t+1]}{\partial \mathbf{s}^N[t]} = 0$.

$(\nabla_{\mathbf{W}^l} L_{sr})_{sr} \neq \mathbf{0}$, we have:

$$
\begin{aligned}
\left\| \widehat{\nabla_{\mathbf{W}^l} L} - (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\| &= \left\| \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \left(\frac{\sum_{\tau=0}^{t-1} \lambda^\tau}{\sum_{\tau=0}^{T-1} \lambda^\tau} \mathbf{a}^l[t]^\top - \mathbf{a}^l[T]^\top \right) \right\| \\
&= \left\| \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \left(\boldsymbol{\epsilon}^l[t] - \frac{\lambda^t (1-\lambda^{T-t})}{1-\lambda^T} \mathbf{a}^l[t]^\top \right) \right\| \\
&\leq \left\| \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \boldsymbol{\epsilon}^l[t] \right\| + \left\| \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \frac{\lambda^t (1-\lambda^{T-t})}{1-\lambda^T} \mathbf{a}^l[t]^\top \right\| \\
&< \left\| \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \mathbf{a}^l[T]^\top \right\| = \left\| (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\|.
\end{aligned}
\tag{9}
$$

Then, we can obtain:

$$
\begin{aligned}
\left\langle \widehat{\nabla_{\mathbf{W}^l} L}, (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\rangle &= \left\langle \widehat{\nabla_{\mathbf{W}^l} L} - (\nabla_{\mathbf{W}^l} L_{sr})_{sr}, (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\rangle + \left\| (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\|^2 \\
&\geq \left\| (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\|^2 - \left\| \widehat{\nabla_{\mathbf{W}^l} L} - (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\| \left\| (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\| > 0.
\end{aligned}
\tag{10}
$$

Therefore, $\left\langle \nabla_{\mathbf{W}^l} L, (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\rangle = \left(\sum_{\tau=0}^{T-1} \lambda^\tau \right) \left\langle \widehat{\nabla_{\mathbf{W}^l} L}, (\nabla_{\mathbf{W}^l} L_{sr})_{sr} \right\rangle > 0$.

□

Remark 1. *As for the assumption of the errors in the theorem, since the weighted firing rate gradually converges $\mathbf{a}[t] \to \mathbf{a}^*$ with bounded random error caused by the remaining membrane potential at the last time step, the order of errors $\boldsymbol{\epsilon}^l[t]$ would be smaller than $\mathbf{a}^l[T]$ especially when t is large. And $\frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T} \to 0$ with $t \to T$ is also a small number on the right side of the inequality. So this is a reasonable assumption.*

Remark 2. *The above conclusion mainly focuses on the gradients for connection weights $\mathbf{W}^l$. As for other parameters such as biases $\mathbf{b}^l$, the gradients of OTTT do not involve pre-synaptic activities, so under Assumption 1 they are exactly the same as gradients based on spike representation except a constant scaling factor $\frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau}$.*

Remark 3. *Note that the gradients based on spike representation may also include small errors since the calculation of SNN is not exactly the same as the equivalent ANN-like mappings. And a larger time step may lead to more accurate gradients. We connect the gradients of OTTT and gradients based on spike representation to demonstrate the overall descent direction, and it is tolerant to small errors, which can also be viewed as randomness for stochastic optimization.*

A.3 Proof of Theorem 2

In this subsection, we prove Theorem 2.

Theorem 2. *If Assumption 1 holds, $V_{th} = 1$, $\left\| J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]} \right\| \leq \eta < \frac{\sigma_{min}^2}{\sigma_{max}^2}$, where σ_{max} and σ_{min} are the maximal and minimal singular value of $\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}|_{\mathbf{a}[T]}$, and the errors $\boldsymbol{\epsilon}^1[t] = \mathbf{a}[t] - \mathbf{a}[T]$, $\boldsymbol{\epsilon}^0[t] = \overline{\mathbf{x}}[t] - \overline{\mathbf{x}}[T]$ are small such that $\left\| \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \boldsymbol{\epsilon}^l[t]^\top \right\| < \frac{\sigma_{min}^2 - \eta \sigma_{max}^2}{\sigma_{max}} \left\| \sum_{t=1}^{T} \frac{\partial \mathcal{L}(\mathbf{s}[t], \mathbf{y})}{\partial \mathbf{s}[t]} \left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]} \right)^{-1} \right\| - \left\| \sum_{t=1}^{T} \frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \mathbf{a}^l[t]^\top \right\|$ (where $l = 0, 1$, $\mathbf{a}^1[t]$ and $\mathbf{a}^0[t]$ represent $\mathbf{a}[t]$ and $\overline{\mathbf{x}}[t]$, respectively) when $(\nabla_{\boldsymbol{\theta}} L_{sr})_{sr} \neq \mathbf{0}$, then we have $\left\langle \nabla_{\boldsymbol{\theta}} L, (\nabla_{\boldsymbol{\theta}} L_{sr})_{sr} \right\rangle > 0$, where $\boldsymbol{\theta}$ are parameters in the network.*

Proof. As described in Sections 4.1 and 4.2 and similar to the proof of Theorem 1, let $\hat{\mathbf{g}}_{\mathbf{u}}[t] = \left(\frac{\partial \mathcal{L}(\mathbf{s}[t], \mathbf{y})}{\partial \mathbf{s}[t]} \frac{\partial \mathbf{s}[t]}{\partial \mathbf{u}[t]} \right)^\top$, we have $\nabla_{\mathbf{W}} L = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \hat{\mathbf{a}}[t]^\top$, $\nabla_{\mathbf{F}} L = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \hat{\mathbf{x}}[t]^\top$ (where $\hat{\mathbf{x}}[t] = \sum_{\tau=1}^{t} \lambda^{t-\tau} \mathbf{x}[\tau]$), and $\nabla_{\mathbf{b}} L = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]$. For gradients based on spike representation, $(\nabla_{\boldsymbol{\theta}} L_{sr})_{sr} = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]} \left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]} \right)^{-1} \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \boldsymbol{\theta}} \right)^\top$, and we will also consider $(\widetilde{\nabla_{\boldsymbol{\theta}} L_{sr}})_{sr} = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]} \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \boldsymbol{\theta}} \right)^\top$. Considering $L_{sr} = \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \sum_{t=1}^{T} \lambda^{T-t} \mathcal{L}(\mathbf{s}[t], \mathbf{y})$, and with Assumption 1 which indicates $\frac{\partial \mathbf{s}_i^{l+1}[t]}{\partial \mathbf{u}_i^{l+1}[t]} = \mathbf{d}_i^{l+1}[T]$, we can derive that $(\widetilde{\nabla_{\mathbf{W}} L_{sr}})_{sr} = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \mathbf{a}[T]^\top$, $(\widetilde{\nabla_{\mathbf{F}} L_{sr}})_{sr} = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t] \overline{\mathbf{x}}[T]^\top$, and $(\widetilde{\nabla_{\mathbf{b}} L_{sr}})_{sr} = \frac{1}{T} \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]$.

We consider $\widehat{\nabla_{\mathbf{W}} L} = \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \nabla_{\mathbf{W}} L = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}^{l+1}}[t] \frac{\sum_{\tau=0}^{t-1} \lambda^\tau}{\sum_{\tau=0}^{T-1} \lambda^\tau} \mathbf{a}^l[t]^\top$ and $\widehat{\nabla_{\mathbf{F}} L} = \frac{1}{\sum_{\tau=0}^{T-1} \lambda^\tau} \nabla_{\mathbf{F}} L$. Since $\left\| J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]} \right\| \leq \eta < \frac{\sigma_{\min}^2}{\sigma_{\max}^2}$, where $\sigma_{\max}$ and $\sigma_{\min}$ are the maximal and minimal singular value of

$\frac{\partial f_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}|_{\mathbf{a}[T]}$ ($\boldsymbol{\theta} \in \{\mathbf{W}, \mathbf{F}, \mathbf{b}\}$), and the errors $\boldsymbol{\epsilon}[t] = \mathbf{a}[t] - \mathbf{a}[T]$ are small such that $\left\|\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\boldsymbol{\epsilon}^{l}[t]^{\top}\right\| < \frac{\sigma_{\min}^2 - \eta\sigma_{\max}^2}{\sigma_{\max}} \left\|\sum_{t=1}^{T} \frac{\partial \mathcal{L}(\mathbf{s}[t],\mathbf{y})}{\partial \mathbf{s}[t]} \left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\right\| - \left\|\sum_{t=1}^{T} \frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\mathbf{a}[t]^{\top}\right\|$ when $\left(\nabla_{\boldsymbol{\theta}} L\right)_{sr} \neq \mathbf{0}$, we can obtain:

$$
\begin{aligned}
\left\|\widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}\right\| &= \left\|\frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\left(\frac{\sum_{\tau=0}^{t-1}\lambda^{\tau}}{\sum_{\tau=0}^{T-1}\lambda^{\tau}}\mathbf{a}[t]^{\top} - \mathbf{a}[T]^{\top}\right)\right\| \\
&= \left\|\frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\left(\boldsymbol{\epsilon}[t] - \frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T}\mathbf{a}[t]^{\top}\right)\right\| \\
&\leq \left\|\frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\boldsymbol{\epsilon}[t]\right\| + \left\|\frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{g}}_{\mathbf{u}}[t]\frac{\lambda^t(1-\lambda^{T-t})}{1-\lambda^T}\mathbf{a}[t]^{\top}\right\| \\
&< \frac{\sigma_{\min}^2 - \eta\sigma_{\max}^2}{\sigma_{\max}}\left\|\frac{1}{T}\sum_{t=1}^{T}\frac{\partial \mathcal{L}(\mathbf{s}[t],\mathbf{y})}{\partial \mathbf{s}[t]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\right\|.
\end{aligned}
\tag{11}
$$

Then, we have (let $\mathbf{v} = \left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\right)^{\top} = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{\partial \mathcal{L}(\mathbf{s}[t],\mathbf{y})}{\partial \mathbf{s}[t]}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{-1}\right)^{\top}$):

$$
\begin{aligned}
\left\langle \widehat{\nabla_{\mathbf{W}} L}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle &= \left\langle \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle + \left\langle \widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle \\
&= \mathbf{v}^{\top}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\left(\frac{\partial L_{sr}}{\partial \mathbf{a}[T]}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\right)^{\top} + \left\langle \widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle \\
&= \mathbf{v}^{\top}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}^{\top}\left(I - J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right)^{\top}\mathbf{v} + \left\langle \widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle \\
&= \left\|\mathbf{v}^{\top}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\right\|^2 - \mathbf{v}^{\top}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}^{\top} J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}{}^{\top}\mathbf{v} \\
&\quad + \left\langle \widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle \\
&\geq \sigma_{\min}^2\|\mathbf{v}\|^2 - \eta\sigma_{\max}^2\|\mathbf{v}\|^2 - \left\|\widehat{\nabla_{\mathbf{W}} L} - \widetilde{(\nabla_{\mathbf{W}} L_{sr})}_{sr}\right\|\left\|\mathbf{v}^{\top}\frac{\partial f_{\boldsymbol{\theta}}(\mathbf{a}[T])}{\partial \mathbf{W}}\right\| \\
&> \sigma_{\min}^2\|\mathbf{v}\|^2 - \eta\sigma_{\max}^2\|\mathbf{v}\|^2 - \frac{\sigma_{\min}^2 - \eta\sigma_{\max}^2}{\sigma_{\max}}\|\mathbf{v}\| \cdot \sigma_{\max}\|\mathbf{v}\| = 0.
\end{aligned}
\tag{12}
$$

Therefore, $\left\langle \nabla_{\mathbf{W}} L, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle = \left(\sum_{\tau=0}^{T-1}\lambda^{\tau}\right)\left\langle \widehat{\nabla_{\mathbf{W}} L}, \left(\nabla_{\mathbf{W}} L_{sr}\right)_{sr}\right\rangle > 0$. Similarly, we can derive that $\left\langle \nabla_{\mathbf{F}} L, \left(\nabla_{\mathbf{F}} L_{sr}\right)_{sr}\right\rangle > 0$. And for $\nabla_{\mathbf{b}} L$, we have $\nabla_{\mathbf{b}} L = \left(\sum_{\tau=0}^{T-1}\lambda^{\tau}\right)\left(\nabla_{\mathbf{b}} L_{sr}\right)_{sr}$, so $\left\langle \nabla_{\mathbf{b}} L, \left(\nabla_{\mathbf{b}} L_{sr}\right)_{sr}\right\rangle > 0$. Therefore, for all parameters $\boldsymbol{\theta}$ in the network, we have $\left\langle \nabla_{\boldsymbol{\theta}} L, \left(\nabla_{\boldsymbol{\theta}} L_{sr}\right)_{sr}\right\rangle > 0$ when $\left(\nabla_{\boldsymbol{\theta}} L_{sr}\right)_{sr} \neq \mathbf{0}$.

□

Remark 4. *The above conclusion considers the single-layer condition. It can be generalized to the multi-layer condition. For example, if we consider multiple feedforward hidden layers (denote the weight as F^l) with a feedback connection from the last hidden layer to the first hidden layer (denote the weight as W^1), and assume the function is contractive, the equilibrium states for each layer are $\mathbf{a}^{1^*} = f_1\left(f_N \circ \cdots \circ f_2(\mathbf{a}^{1^*}), \mathbf{x}^*\right)$ and $\mathbf{a}^{l+1^*} = f_{l+1}(\mathbf{a}^{l^*})$, where $f_1(\mathbf{a}, \mathbf{x}) = \sigma\left(\frac{1}{V_{th}}(\mathbf{W}^1\mathbf{a} + \mathbf{F}^1\mathbf{x} + \mathbf{b}^1)\right)$ and $f_l(\mathbf{a}) = \sigma\left(\frac{1}{V_{th}}(\mathbf{F}^l\mathbf{a} + \mathbf{b}^l)\right)$ [12]. Then with a similar condition for the Jacobian of $f_{\boldsymbol{\theta}} = f_N \circ \cdots \circ f_2 \circ f_1$ and errors $\boldsymbol{\epsilon}^l[t]$ of each layer as in Theorem 2, we can prove $\left\langle \nabla_{\boldsymbol{\theta}} L, \left(\nabla_{\boldsymbol{\theta}} L\right)_{sr}\right\rangle > 0$ when $\left(\nabla_{\boldsymbol{\theta}} L\right)_{sr} \neq \mathbf{0}$ for all parameters $\boldsymbol{\theta}$ in the network as well. More generally, multi-layer networks with arbitrary feedback connections can be written in a single-layer formulation, i.e. we consider all neurons in different layers as a whole single layer, and feedforward or feedback connections can be viewed as connections between these neurons, which is written as a much larger weight matrix with some imposed structures representing the connection restrictions. Therefore, the conclusion can be directly generalized to these conditions as well.*

Remark 5. *The assumption $\left\|J_{f_{\boldsymbol{\theta}}}|_{\mathbf{a}[T]}\right\| \leq \eta < \frac{\sigma_{min}^2}{\sigma_{max}^2}$ is also made in previous works [35, 36] and we consider it as a reasonable assumption in the theoretical analysis. It is a sufficient condition to bound the worst case, and in practice it is unnecessary to always enforce the restriction, as indicated in [35].*

B Pseudocode of the OTTT algorithm

We present the pseudocode of one iteration of OTTT training for a feedforward network in Algorithm 1 to better illustrate our training method.

Algorithm 1 One iteration of OTTT training for a feedforward network.

Input: Network parameters $\{\mathbf{W}^l\}, \{\mathbf{b}^l\}$; Input data x; Label y; Time steps T; Other hyperparameters;
Output: Trained network parameters $\{\mathbf{W}^l\}, \{\mathbf{b}^l\}$.

for $t = 1, 2, \cdots, T$ **do**
  for $l = 1, 2, \cdots, N$ **do** // **Forward**
    Update membrane potentials $\mathbf{u}^l[t]$ and generate spikes $\mathbf{s}^l[t]$ at layer l;
    Update the tracked presynaptic activities $\hat{\mathbf{a}}^l[t] = \lambda \hat{\mathbf{a}}^l[t-1] + \hat{\mathbf{s}}^l[t]$ at layer l.
  for $l = N, N-1, \cdots, 1$ **do** // **Backward**
    Calculate the instantaneous backpropagated errors $\mathbf{g}_{\mathbf{u}^l}[t]$;
    Calculate the instantaneous gradient $\nabla_{\mathbf{W}^{l-1}} L[t] = \mathbf{g}_{\mathbf{u}^l}[t](\hat{\mathbf{a}}^{l-1}[t])^\top$.
    if online update **then** // OTTT$_O$
      Update $\mathbf{W}^{l-1}$ with $\nabla_{\mathbf{W}^{l-1}} L[t]$ based on the gradient-based optimizer;
      Update $\mathbf{b}^l$ with $\mathbf{g}_{\mathbf{u}^l}[t]$ based on the gradient-based optimizer.
    else // OTTT$_A$
      Accumulate gradients $\nabla_{\mathbf{W}^{l-1}} L = \nabla_{\mathbf{W}^{l-1}} L + \nabla_{\mathbf{W}^{l-1}} L[t]$, $\nabla_{\mathbf{b}^l} L = \nabla_{\mathbf{b}^l} L + \mathbf{g}_{\mathbf{u}^l}[t]$.
if not online update **then** // OTTT$_A$
  Update parameters $\{\mathbf{W}^l\}$ with accumulated $\{\nabla_{\mathbf{W}^l} L\}$ based on the gradient-based optimizer;
  Update parameters $\{\mathbf{b}^l\}$ with accumulated $\{\nabla_{\mathbf{b}^l} L\}$ based on the gradient-based optimizer.

C Implementation Details

C.1 Scaled Weight Standardization and NF-ResNets

The scaled weight standardization (sWS) is proposed in [54, 55] to replace the commonly used batch normalization (BN) and realize normalization-free ResNets (NF-ResNets). Different from BN which standardizes the activation with different samples, sWS standardizes weights by:

$$\hat{\mathbf{W}}_{i,j} = \gamma \cdot \frac{\mathbf{W}_{i,j} - \mu_{\mathbf{W}_{i,\cdot}}}{\sigma_{\mathbf{W}_{i,\cdot}} \sqrt{N}}, \tag{13}$$

where $\mu_{\mathbf{W}_{i,\cdot}}$ and $\sigma_{\mathbf{W}_{i,\cdot}}$ are the mean and variance calculated along the input dimension, and the scale γ is determined by analyzing the signal propagation with different activation functions. The original weight standardization is proposed in [56], which is shown to share the similar benefit as BN to smooth the loss landscape, if combined with other normalization techniques, e.g. group normalization. sWS further takes the signal propagation into account so that the variance of the signal is preserved during the forward propagation of neural networks and the mean of the output is 0, which is another property of BN. Particularly, for the input $\mathbf{x}$ that is sampled i.i.d from $\mathcal{N}(0, 1)$, considering the ReLU activation g, [54] derive that we should take $\gamma = \frac{\sqrt{2}}{\sqrt{1-\frac{1}{\pi}}}$ to preserve the variance of signals, i.e. $\mathrm{Var}(\hat{\mathbf{W}} g(\mathbf{x})) = 1$. This is because the outputs $g(x) = \max(x, 0)$ with Gaussian inputs will be sampled from the rectified Gaussian distribution with variance $\sigma_g^2 = (1/2)(1 - (1/\pi))$ [54]. In this work, to ensure the variance preserving at each time step of the SNN computation, we derive γ based on the consideration of the signals after the Heaviside step function H. Particularly, consider the Gaussian input $\mathbf{x}$, when $V_{th} = 1$, the variance of the outputs $H(x - V_{th})$ is $\sigma_H^2 = \frac{1}{2}\mathrm{erfc}(\frac{1}{\sqrt{2}}) \left(1 - \frac{1}{2}\mathrm{erfc}(\frac{1}{\sqrt{2}})\right)$. So we will take $\gamma = \frac{1}{\sigma_H} \approx 2.74$ to preserve the variance of signals. Additionally, [54] demonstrates that sWS can incorporate another learnable scaling factor for the weights, which is also taken in common BN implementations. Therefore, we also adopt this sWS technique, which is the same as the pseudocode in [54]. For VGG network structures, we directly impose sWS on all weights. For NF-ResNet structures, we use the same structure as in [54], which is briefly introduced below.

NF-ResNets [54] consider the residual networks $x_{l+1} = x_l + \alpha f_l(x_l/\beta_l)$, which differs from ResNets [64] in three aspects: 1. NF-ResNets remove the BN components in ResNets and impose sWS on all weights; 2. a scaling factor α is added for each residual branch; 3. for the input of each residual branch, it will first be divided by the term β_l that represents the standard deviation of signals. Note that the third point is because the

residual computation $x_{l+1} = x_l + \alpha f_l(x_l/\beta_l)$ will gradually accumulate the variance of the residual branch, i.e. $\text{Var}(x_{l+1}) = \text{Var}(x_l) + \text{Var}(\alpha f_l(x_l/\beta_l))$, so dividing β_l ensures that the residual branch keeps the identity variance 1 (combined with sWS), and this also indicates to calculate β_l by $\beta_{l+1}^2 = \beta_l^2 + \alpha^2$ after each branch. Also, note that for each transition block, the identity path with a strided conv will also be first divided by β_l, so the variance is reset after each transition block between two stages. For the implementation details, we mainly follow the pseudocode in [54] and replace the activation functions by functions of spiking neurons, and we take $\alpha = 0.2$. For more illustrations and other details, please directly refer to [54].

C.2 Training Settings

C.2.1 Datasets

We conduct experiments on CIFAR-10 [58], CIFAR-100 [58], ImageNet [59], CIFAR10-DVS [60], and DVS128-Gesture [61].

CIFAR-10 CIFAR-10 is a dataset of color images with 10 classes of objects, which contains 50,000 training samples and 10,000 testing samples. Each sample is a $32 \times 32 \times 3$ color image. We normalize the inputs based on the global mean and standard deviation, and apply random cropping, horizontal flipping and cutout [65] for data augmentation. The inputs to the first layer of SNNs at each time step are directly the pixel values, which can be viewed as a real-valued input current.

CIFAR-100 CIFAR-100 is a dataset similar to CIFAR-10 except that there are 100 classes of objects. It also consists of 50,000 training samples and 10,000 testing samples. We use the same pre-processing as CIFAR-10.

The license of CIFAR-10 and CIFAR-100 is the MIT License.

ImageNet ImageNet-1K is a dataset of color images with 1000 classes of objects, which contains 1,281,167 training samples and 50,000 validation images. We adopt the common pre-possessing strategies, i.e. the training images are first randomly resized and cropped to 224×224, and then normalized after the random horizontal flipping data augmentation, while the testing images are first resized to 256×256 and center-cropped to 224×224, and then normalized. The inputs are also converted to a real-valued input current at each time step. The license of ImageNet is Custom (non-commercial).

DVS-CIFAR10 The DVS-CIFAR10 dataset is the neuromorphic version of the CIFAR-10 dataset converted by a Dynamic Vision Sensor (DVS), which is composed of 10,000 samples, one-sixth of the original CIFAR-10. It consists of spike trains with two channels corresponding to ON- and OFF-event spikes. The pixel dimension is expanded to 128×128. Following the common practice, we split the dataset into 9000 training samples and 1000 testing samples. As for the data pre-processing, we reduce the time resolution by accumulating the spike events [23] into 10 time steps, and we reduce the spatial resolution into 48×48 by interpolation. We apply the random cropping augmentation as CIFAR-10 to the input data, and normalize the inputs based on the global mean and standard deviation of all time steps (which can be integrated into the connection weights of the first layer). The license of DVS-CIFAR10 is CC BY 4.0.

DVS128-Gesture The DVS128-Gesture dataset is a neuromorphic dataset that contains 11 kinds of hand gestures from 29 subjects under 3 kinds of illumination conditions recorded by a DVS camera. It is composed of 1176 training samples and 288 testing samples. Following [23], we pre-possess the data to integrate event data into 20 frames. The license of DVS128-Gesture is the Creative Commons Attribution 4.0 license.

C.2.2 Training Hyperparameters

For our SNN models, we assume the neurons of the last classification layer will not spike or reset, and do classification based on the accumulated membrane potential, which is the same as [12]. That is, the final output is $\mathbf{u}^N[t] = \mathbf{W}^{N-1}\mathbf{s}^{N-1}[t] + \mathbf{b}^N$ at each time step. The classification is based on the accumulated $\mathbf{u}^N = \sum_{t=1}^{T} \mathbf{u}^N[t]$, and the loss during training is also calculated based on $\mathbf{u}^N[t]$, i.e. $\mathcal{L}(\mathbf{u}^N[t], \mathbf{y})$.

For CIFAR-10, CIFAR-100, and DVS-CIFAR10, models are trained by SGD with momentum 0.9 for 300 epochs with the default batch size 128, and the initial learning rate is set as 0.1 with a cosine annealing learning rate scheduler to 0 (for the experiments of training with batch size 1, the initial learning rate is linearly rescaled to $\frac{0.1}{128}$). For DVS-CIFAR10, we apply dropout on all layers with dropout rate as 0.1. As for the loss function, inspired by [13], we combine cross-entropy (CE) loss and mean-square-error (MSE) loss, i.e. $\mathcal{L}(\mathbf{u}^N[t], \mathbf{y}) = (1-\alpha)\mathbf{CE}(\mathbf{u}^N[t], \mathbf{y}) + \alpha\mathbf{MSE}(\mathbf{u}^N[t], \mathbf{y})$, where α is taken as 0.05 for CIFAR10 and CIFAR100 while 0.001 for DVS-CIFAR10.

For ImageNet, models are trained by SGD with momentum 0.9 for 100 epochs with the default batch size 256, and the initial learning rate is set as 0.1, which is decayed by 0.1 every 30 epochs. We set the weight decay as 2×10^{-5}, and no dropout is applied. The loss function takes the cross-entropy loss.

For DVS128-Gesture, models are trained by the Adam optimizer for 300 epochs with batch size 16, and the initial learning rate is set as 0.001 with a cosine annealing learning rate scheduler to 0. No dropout is applied. As for the loss function, we set $\alpha = 0.001$ following DVS-CIFAR10.

The code implementation is based on the PyTorch framework [66], and experiments are carried out on one NVIDIA GeForce RTX 3090 GPU.

D Additional Experiment Results

D.1 Firing Rate Statistics on ImageNet

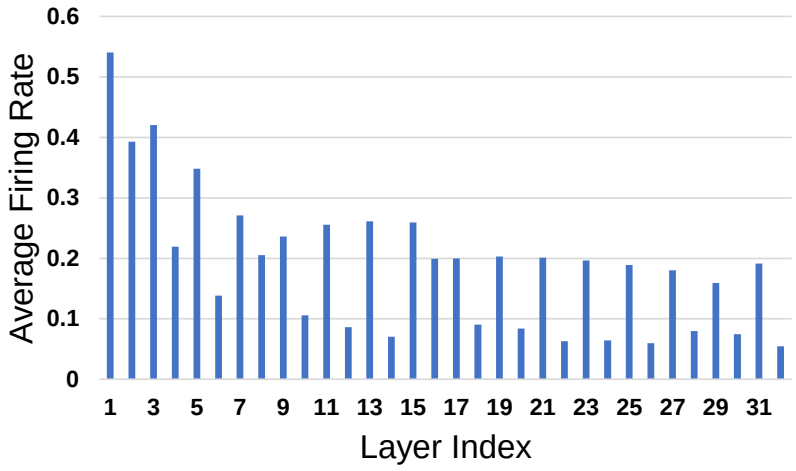


Figure 5: The average firing rates for the model trained by OTTT_A on ImageNet.

In this section, we supplement the firing rate statistics of the NF-ResNet-34 model trained by OTTT_A on ImageNet, as shown in Fig. 5. Overall the firing rate is around 0.24 and with 6 time steps each neuron averagely generate 1.46 spikes. Note that we can also reduce the time steps to realize a trade-off between accuracy and energy, as shown in Fig. 3 in Section 5.5. For example, with 2 time steps each neuron only averagely generate 0.48 spikes, with around 2.5% accuracy drop.

D.2 Comparison between OTTT and BPTT with Feedback Connections

In this section, we supplement the results to compare the performance of OTTT and BPTT with feedback connections. As shown in Table 5, feedback connections can improve the performance for both OTTT and BPTT, and the improvement of OTTT from feedback connections is more significant than that of BPTT.

Table 5: Performance on CIFAR-100 for VGG and VGG-F trained by OTTT_O and BPTT.

Method	Network structure	Params	Mean$\pm$Std (Best)
OTTT_O (ours)	VGG	9.3M	71.05$\pm$0.06% (71.11%)
OTTT_O (ours)	VGG-F	9.6M	72.63$\pm$0.23% (72.94%)
BPTT	VGG	9.3M	69.06$\pm$0.07% (69.15%)
BPTT	VGG-F	9.6M	(69.49%)

D.3 Experiments on Fully Recurrent Structures

In this section, we supplement an experiment to use a recurrent spiking neural network to classify the Fashion-MNIST dataset [67]. The input is flattened as a vector with 784 dimensions, and is connected to 400 spiking neurons with recurrent connections, which are then connected to a readout layer for classification. We apply

weight standardization for connection weights from inputs to hidden neurons. Models are trained by 100 epochs with batch size 128 and SGD with momentum 0.9. The initial learning rate is set as 0.1 with a cosine annealing learning rate scheduler to 0. Dropout is set as 0.2, and weight decay is set as 5e-4 for BPTT and OTTT$_A$ while 1e-4 for OTTT$_O$ (since OTTT$_O$ update more times for each iteration). As for the loss function, we set $\alpha = 0.05$ following CIFAR-10. As shown in Table 6, for this relatively simple model, the results of OTTT and BPTT are similar and BPTT performs slightly better.

Table 6: Performance on Fashion-MNIST.

Method	Network structure	Time steps	Accuracy
ST-RSBP [68]	400 (R400)	400	90.00±0.14% (90.13%)
IDE [12]	400 (R400)	5	90.07±0.10% (90.25%)
BPTT	400 (R400)	5	90.58%
OTTT$_A$ (ours)	400 (R400)	5	90.36%
OTTT$_O$ (ours)	400 (R400)	5	90.40%

E Discussion of Limitations and Social Impacts

This work focus on online training of spiking neural networks, and therefore limits the usage of some techniques on network structures such as batch normalization along the temporal dimension. In this work, we adopt the scaled weight standardization as an alternative, which may require additional regularization to fully catch up the best performance of batch normalization as shown in the results of ANNs [54]. It may require exploration of more techniques that is specific for SNNs to improve the performance and meanwhile compatible with more natural properties of SNNs, e.g. the online property.

As for social impacts, since this work focuses only on training methods for spiking neural networks, there is no direct negative social impact. And we believe that the development of successful energy-efficient SNN models could broader its applications and alleviate the huge energy consumption by ANNs. Besides, understanding and improving the training of biologically plausible SNNs may also contribute to the understanding of our brains and bridge the gap between biological neurons and successful deep learning.