NDOT: Neuronal Dynamics-based Online Training for Spiking Neural Networks

Haiyan Jiang [1] **Giulia De Masi** [2 3] **Huan Xiong** [1 4] **Bin Gu** [1 5]

Abstract

Spiking Neural Networks (SNNs) are attracting great attention for their energy-efficient and fast-inference properties in neuromorphic computing. However, the efficient training of deep SNNs poses challenges in *gradient calculation* due to the non-differentiability of their binary spike-generating activation functions. To address this issue, the surrogate gradient (SG) method is widely used, typically in combination with back-propagation through time (BPTT). Yet, BPTT's process of unfolding and back-propagating along the computational graph requires storing intermediate information at all time-steps, resulting in huge memory consumption and unable to meet online requirements. In this work, we propose Neuronal Dynamics-based Online Training (NDOT) for SNNs, which uses the neuronal dynamics-based *continuous temporal dependency* in *gradient computation*. NDOT enables ***forward-in-time learning*** by decomposing the full gradient into temporal and spatial gradients. To illustrate the intuition behind NDOT, we employ the Follow-the-Regularized-Leader (FTRL) algorithm. FTRL *explicitly* utilizes historical information and addresses limitations in instantaneous loss. Our proposed NDOT method uses neuronal dynamics to accurately capture *temporal dependencies*, functioning similarly to FTRL's explicit use of historical information. Experiments on CIFAR-10, CIFAR-100, and CIFAR10-DVS demonstrate the superior performance of our NDOT method on large-scale static and neuromorphic datasets within a small number of time steps. The codes are available at https://github.com/HaiyanJiang/SNN-NDOT.

[1]Department of Machine Learning, MBZUAI, Abu Dhabi, UAE [2]Technology Innovation Institute, Abu Dhabi, UAE [3]Sant'Anna School of Advanced Studies, Italy [4]Harbin Institute of Technology, China [5]School of Artificial Intelligence, Jilin University, China. Correspondence to: Huan Xiong <Huan.Xiong@mbzuai.ac.ae>, Bin Gu <Bin.Gu@mbzuai.ac.ae>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

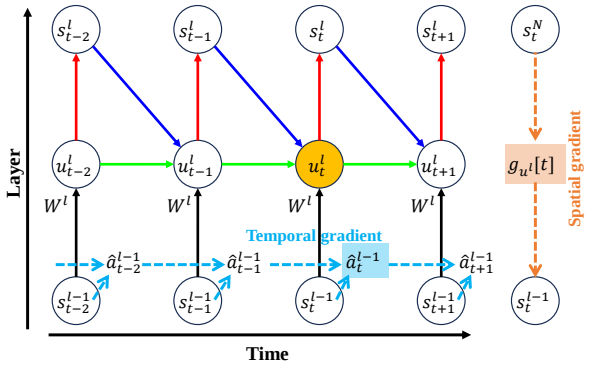

Figure 1. Illustration of the computational graph of multi-layer SNNs and the full gradient calculation process of NDOT. The full gradient calculation includes the temporal gradient calculation and spatial gradient calculation.

1. Introduction

Spiking Neural Networks (SNNs), as the third generation of neural networks (Maass, 1997; Roy et al., 2019), offer a more biologically plausible model when compared to their second-generation counterparts, Artificial Neural Networks (ANNs). SNNs emulate the dynamics of biological neurons, communicating between layers through spike trains (i.e., time series of spikes). Recently, SNNs have gained increasing research attention (Lee et al., 2016; Shrestha & Orchard, 2018; Xiao et al., 2021; Fang et al., 2021a; Jiang et al., 2023; Deng et al., 2023), due to their energy-efficient and fast-inference capabilities on neuromorphic hardware like TrueNorth (Akopyan et al., 2015), Loihi (Davies et al., 2018) and Tianjic (Pei et al., 2019). These advantages arise from the binary, event-driven nature of spikes, which allows SNNs to bypass multiplication during inference. However, while the binary spiking signals offer SNNs high speed and low energy consumption, the discontinuity caused by the binary spike-generation process poses challenges for the supervised training of SNNs from scratch (Wu et al., 2018; Deng et al., 2021; Meng et al., 2022; Deng et al., 2023).

The training algorithms for deep SNNs are generally divided into two categories: ANN-to-SNN conversion and gradient-based direct training. In the ANN-to-SNN conversion approach, an SNN is obtained from a pre-trained ANN, by transferring/converting network weights from an equivalent ANN counterpart which is easier to train. Sev-

eral studies have adopted this strategy by first training an ANN and then converting it to an SNN (Diehl et al., 2015; Deng & Gu, 2020; Li et al., 2021a; Bu et al., 2021; Jiang et al., 2023). However, these methods often require longer simulation time-steps to achieve comparable performance to ANNs, leading to increased latency and energy consumption. Additionally, the conversion method may not be well suited for neuromorphic data.

A commonly used direct training method for SNNs is to utilize surrogate gradients (SG) to address the non-differentiability problem caused by the binary spike generation activation functions (Neftci et al., 2019; Zheng et al., 2021). In the forward pass, the Heaviside function is used to generate spikes, while in the backward pass, the non-differentiable Heaviside function is replaced by a differentiable surrogate function. The surrogate gradient (SG) approach involves treating the SNN as a binary Recurrent Neural Network (RNN) and applying backpropagation through time (BPTT) (Neftci et al., 2019; Bellec et al., 2018; Zenke & Ganguli, 2018; Wu et al., 2018). These methods treat SNNs as RNNs, requiring them to store intermediate information of the unrolled computation graph for all time-steps during backpropagation. This need to retain the state of each time-step results in a large memory consumption, which grows linearly with the number of simulated time-steps. This significant training cost is particularly pronounced for large-scale datasets like ImageNet.

Another direct training approach for SNNs is to use spike representations, usually in the form of (weighted) firing rates (Xiao et al., 2021; Meng et al., 2022). Spike representations consolidate information across the temporal dimension of neurons, obviating the need for explicitly considering the temporal dimension during SNN training. By associating SNN spike representations with equivalent ANN-like mappings (Thiele et al., 2019; Wu et al., 2021; Zhou et al., 2021; Xiao et al., 2021; Meng et al., 2022), these methods effectively sidestep the requirement for unrolling and backpropagating through the time dimension in SNNs training. As a result, SNNs can be trained similarly to conventional ANNs. In this process, the spike representations function like activation values, enabling gradients to be calculated based on the corresponding mappings between different layers of these spike representations. However, these methods typically need more time-steps to achieve competitive performance, resulting in increased latency and increased energy consumption when using rate-based representation.

Recent advances in online training techniques strive to reduce memory costs throughout the training process while preserving the biologically plausible online learning properties (Xiao et al., 2022; 2021; Meng et al., 2023). However, *the approximation of temporal dependencies* brings great challenges for accurate gradient calculation. In this work, we introduce a forward-in-time learning method called Neuronal Dynamics-based Online Training (NDOT) to directly train SNNs to achieve high performance with low latency while maintaining the online learning properties. NDOT is derived from the widely used BPTT method, with a detailed analysis of the temporal dependencies. The key idea of NDOT is to effectively represent the temporal dependencies through Neuronal Dynamics. By tracking the temporal gradients forward and incorporating instantaneous loss, NDOT enables forward-in-time online learning, ensuring that gradient calculation is performed online in real-time without the need for backward computation through the time dimension.

Our main contributions can be summarized as follows:
(a) We propose *Neuronal Dynamics-based Online Training* to directly train SNNs, which enables forward-in-time learning without requiring large amounts of training memory. NDOT ensures constant training memory, independent of the time-steps.
(b) *Accurate Temporal Gradient Capture*: Our NDOT accurately captures the temporal gradients along the temporal dimension using Neuronal Dynamics in spiking neurons. This, combined with spatial gradients across layers, allows for straightforward derivation of full gradients, facilitating efficient forward-in-time learning.
(c) *Intuitive Insights Gained through the Follow-the-Regularized-Leader (FTRL) Algorithm:* We explain the intuition behind NDOT with the help of FTRL and demonstrate how NDOT implicitly captures historical information, similar to FTRL's explicit use of historical information.
(d) *Superior Performance on Different Datasets:* Experimental results on CIFAR-10, CIFAR-100, and CIFAR10-DVS demonstrate the outstanding performance of NDOT on large-scale static and neuromorphic datasets within a small number of time-steps.

2. Related Work

BPTT Framework for SNNs. A commonly used methodology for supervised direct training SNNs is to follow the BPTT framework and deal with the non-differentiable problem of spiking generation functions by applying surrogate gradients (SG) to facilitate meaningful gradient calculation (Wu et al., 2018; Huh & Sejnowski, 2018; Shrestha & Orchard, 2018; Neftci et al., 2019; Wu et al., 2019; Zenke & Vogels, 2021). A tremendous amount of novel techniques have been proposed to enhance performance. These include threshold-dependent batch normalization (Zheng et al., 2021), temporal effective batch normalization (Duan et al., 2022), carefully designed differentiable surrogate functions (Li et al., 2021b), different loss functions (Deng et al., 2021; Guo et al., 2022), trainable neuronal parameters (Fang et al., 2021b), and specialized network structures for SNNs (Fang et al., 2021a). Several studies use

multi-stage training, typically involving an initial ANN pre-training phase, to address the energy efficiency concern by reducing the latency (i.e., the number of time-steps) while maintaining high performance (Chowdhury et al., 2022; Rathi & Roy, 2021; Rathi et al., 2019). The BPTT with SG method has demonstrated competitive performance with extremely low latency on both static and neuromorphic datasets (Deng et al., 2021; Fang et al., 2021a; Li et al., 2022). However, the BPTT training requires significant memory and computing resources to backpropagate signals (through both temporal and spatial domains), because SNNs are treated as RNNs and the computational graph needs to be unfolded along the time dimension (Deng et al., 2020; Meng et al., 2023; Xiao et al., 2022). Furthermore, these methods cannot realize online learning, particularly forward-in-time learning which is crucial for neuromorphic hardware compatibility. In contrast, our NDOT method reduces the huge memory cost of the BPTT framework and maintains the online property.

Online Training Neural Networks. Our focus is on algorithms that process samples sequentially to facilitate real-time parameter updates, which are particularly useful for RNNs and SNNs across multiple time-steps. In the domain of RNNs, various alternatives of BPTT have been proposed to enable online learning, such as Real-Time Recurrent Learning (RTRL) (Williams & Zipser, 1989), UORO (Tallec & Ollivier, 2018), KF-RTRL (Mujika et al., 2018), and SnAp (Menick et al., 2020). Particularly, the Forward Propagation Through Time (FPTT), introduced by (Kag & Saligrama, 2021), updates parameters in an online fashion, using decoupled gradients in combination with regularization at each time-step. Yet, these methods are designed for RNNs and are not tailored to SNNs. In the domain of SNNs, numerous studies have drawn inspiration from online training techniques developed for RNNs. Several online training techniques have been introduced and adapted for SNNs (Zenke & Ganguli, 2018; Bellec et al., 2020; Bohnstingl et al., 2022), adopting similar ideas to achieve memory-efficient and online learning. In Kaiser et al. (2020), local losses are leveraged, and temporal dependencies are disregarded for the online local training of SNNs. Meanwhile, (Yin et al., 2023) directly applies the method introduced in (Kag & Saligrama, 2021) for SNN training, requiring a specially designed gated neuron model. Online Training Through Time (OTTT) (Xiao et al., 2022) successfully extends online training methods to handle large-scale tasks such as the ImageNet classification. Recently, the Spatial Learning Through Time (SLTT) (Meng et al., 2023) was introduced for SNNs, arguing that gradients through the temporal domain are unimportant and should be ignored in the computational graph during backpropagation. However, these approaches for SNNs do not incorporate precise gradients along the temporal domain, leading to a performance disadvantage due to gradient approximation compared to their BPTT counterparts in RNNs.

Regularized Online Optimization. Online algorithms operate by drawing random examples one at a time and adjusting the learning variables based on observations that are currently available (Xiao, 2009). We introduce *regularized* stochastic learning and *online optimization* problems, where samples are processed sequentially as they become available (Shalev-Shwartz et al., 2012). There is a lot of research work on regularized online optimization, with most focusing on promoting sparsity (Duchi et al., 2011; Xiao, 2009; Duchi & Singer, 2009). Without regularization, online gradient descent updates the parameters in real-time by subtracting the gradient of the instantaneous loss $\ell_t(w)$ incurred in round t from the current parameter w_t, i.e., $w_{t+1} = w_t - \eta \nabla_w \ell_t(w)$. With a "hard" set constraint (the regularization), one of the most widely used online algorithms is the Online Stochastic Gradient Descent (OSGD) method (Robbins & Monro, 1951; Kiefer & Wolfowitz, 1952) which updates $w_{t+1} = \prod_{\mathcal{C}}(w_t - \eta_t \nabla_w \ell_t(w))$, where $\prod_{\mathcal{C}}$ denotes Euclidean projection onto the set $\mathcal{C}$ and η_t is an appropriate stepsize. The OSGD method has been very popular in the machine learning community due to its capability of scaling with very large data sets and good generalization performances observed in practice (Bottou & Bousquet, 2007; Langford et al., 2009; Shalev-Shwartz et al., 2007). For example, Truncated Gradient (TG) was proposed by (Langford et al., 2009) to induce sparsity in the weights for online learning with convex loss functions. Regularized Dual Averaging (RDA) method (Xiao, 2009) was introduced to exploit the regularization structure in an online setting. At each iteration of RDA methods, the learning variables are adjusted by solving a simple minimization problem that involves *the running average* of all past subgradients of the loss function and *the whole regularization term*, not just its subgradient. At the same time, the FOBOS (Duchi & Singer, 2009) framework was proposed for empirical loss minimization with regularization, which alternates between two phases. On each iteration, an unconstrained gradient descent step is first performed, then an instantaneous optimization problem is cast and solved which trades off minimization of a regularization term while keeping close proximity to the result of the first phase. More descent algorithms were developed such as Composite Objective Mirror Descent (COMID) algorithm (Duchi et al., 2010), Adaptive Online Gradient Descent (AOGD) algorithm (Hazan et al., 2007). Finally, it is proved that many mirror descent algorithms for online convex optimization (such as online gradient descent) have an equivalent interpretation as follow-the-regularized-leader (FTRL) algorithms (McMahan, 2011), where the key difference between these algorithms is how they handle the cumulative L_1-penalty.

3. Preliminaries

3.1. Spiking Neural Networks

As the foundational elements within SNNs, spiking neurons draw inspiration from the complex functions of biological neurons in the brain. In SNNs, a spiking neuron integrates input spike trains into its membrane potential, $u(t)$, and fires a spike only when $u(t)$ exceeds a threshold, and then resets its membrane potential $u(t)$ after firing (Gerstner et al., 2014). Spikes are numerically expressed as 0 and 1, indicating resting and active, respectively. This 0-1 spike train conveys network information between layers, enabling energy-efficient and event-based computation on neuromorphic chips (Davies et al., 2018; Pei et al., 2019).

The leaky integrate-and-fire (LIF) neuron model is a widely used spiking neuron model, which captures the neuronal dynamics of the membrane potential as follows:

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

where τ is the time constant, $I(t)$ is the input current, and V_{th} is the threshold. A spike is generated when $u(t)$ reaches V_{th} at time t_f, and $u(t)$ is reset to the resting potential $u(t) = u_{rest}$, which is often zero. The spike train is defined using the Dirac delta function: $s(t) = \sum_{t_f} \delta(t - t_f)$.

Spiking neural networks consist of multiple layers with interconnected neurons and associated connection weights. Consider a simple input current model $I_i[t] = \sum_j W_{ij} s_j[t]$, where $I_i[t]$ is the current at time-step t for neuron i, W_{ij} represents the weight from neuron j in the previous layer to neuron i in the current layer. To establish the computational link between layers and time-steps along the spatial-temporal dimension, the discrete LIF model can be formulated as:

$$u_i[t+1] = \lambda\left(u_i[t] - V_{th} s_i[t]\right) + \sum_j W_{ij} s_j[t+1]\ , \quad (2)$$

$$s_i[t] = H(u_i[t] - V_{th})\ , \quad (3)$$

where $H(\cdot)$ is the Heaviside step function, $s_i[t]$ is the spike train at discrete time-step $[t]$ for neuron i, and λ is a leaky constant (typically taken as $1 - \frac{1}{\tau}$). The reset operation is implemented by subtraction the threshold V_{th}.

3.2. Previous SNN Training Methods

Spike Representation. The spike representation (Xiao et al., 2021; Meng et al., 2022), i.e., (weighted) firing rate, merges information across the temporal dimension of neurons, eliminating the need to explicitly backpropagating through the temporal dimension during SNN training. The spike representation makes training SNNs feasible by calculating the gradients from the equivalent mappings between spike representations, similar to the training process for ANNs.

We focus on the weighted firing rate which has connections with NDOT in this work. Define the weighted firing rates $\mathbf{a}[t] = \frac{\sum_{\tau=1}^{t} \lambda^{t-\tau} \mathbf{s}[\tau]}{\sum_{\tau=1}^{t} \lambda^{t-\tau}}$, the weighted average inputs $\bar{\mathbf{x}}[t] = \frac{\sum_{\tau=1}^{t} \lambda^{t-\tau} \mathbf{x}[\tau]}{\sum_{\tau=1}^{t} \lambda^{t-\tau}}$ in the discrete condition. With multi-layer feed-forward SNNs, the closed-form mappings between successive layers can be established with the weighted firing rates, $\mathbf{a}^l[T] \approx \sigma\left(\frac{1}{V_{th}} \mathbf{W}^l \mathbf{a}^{l-1}[T]\right)$, where $\sigma(x) = \min(\max(0, x), 1)$ is a clamp function in discrete version. The gradients can be calculated with this spike representation $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^N[T]} \prod_{j=l}^{N-1} \frac{\partial \mathbf{a}^{j+1}[T]}{\partial \mathbf{a}^j[T]} \frac{\partial \mathbf{a}^l[T]}{\partial \mathbf{W}^l}$, where the weighted firing rates $\mathbf{a}^l[T]$, i.e. spike representations, are similar to the activations between layers in ANNs.

BPTT (backpropagation through time) with SG. Consider the multi-layer feedforward SNNs with the LIF neuron model based on Eq. (2)

$$\mathbf{u}^l[t+1] = \lambda\left(\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]\right) + \mathbf{W}^l \mathbf{s}^{l-1}[t+1]\ , \quad (4)$$

where $l \in \{1, \cdots, N\}$ is the layer index, $t \in \{1, \cdots, T\}$ is the discrete time-step index, $\mathbf{s}^l[t]$ are the output spike trains of the l-th layer, $\mathbf{W}^l$ are the weights to be trained, and $\mathbf{s}^N[t]$ are the output spike trains at the last layer.

The BPTT method unfolds the iterative update equation in Eq. (4) and backpropagates through time along the unfolded computational graph as shown in Fig. 1. The gradients with T time-steps are calculated by [1]

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left\{ \frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} \right. \quad (5)$$
$$\left. + \sum_{k<t} \prod_{i=k}^{t-1} \left(\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{u}^l[i]} + \frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]} \right) \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right\},$$

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

3.3. Follow-the-Regularized-Leader (FTRL)

In *regularized* stochastic learning and *online* optimization problems, the objective function $f_t(\cdot)$ is the sum of two convex terms: the loss function of the learning task $\ell_t(\cdot)$, and the regularization function of exploiting problem structure $\Psi(\cdot)$, i.e., $f_t(w) = \ell_t(w) + \Psi(w)$. The L_1-regularization is popularly used for promoting sparsity. Online algorithms process samples sequentially as they become available. More specifically, on each round $t = 1, 2, \cdots, T$, we draw a sequence

[1] This paper follows numerator layout convention. $\nabla_{\mathbf{W}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}}\right)^\top$.

of i.i.d. samples $z_1, z_2, \cdots, z_T$ and use them to calculate a sequence of weights $w_1, w_2, \cdots, w_T$. Suppose at time t, we have the most up-to-date weight vector w_t. Whenever z_t is available, we can evaluate the instantaneous loss $\ell_t(w_t; z_t)$ and also a subgradient $g_t = \nabla \ell_t(w_t; z_t)$ (with respect to w). Then we compute w_{t+1} based on this information.

The follow-the-regularized-leader (FTRL) algorithm is the follow-the-leader algorithm with a regularized term $\Psi(\cdot)$ added, where $\Psi : W \to \mathbb{R}$. Specifically, for L_1-regularization, i.e., $\Psi(w) = \lambda \|w\|_1$, the FTRL is highly effective in obtaining sparse solutions. Numerous mirror descent algorithms (McMahan & Streeter, 2010; Duchi et al., 2011), including RDA (Xiao, 2009), FOBOS (Duchi & Singer, 2009), AOGD (Hazan et al., 2007), are demonstrated to have an equivalent interpretation within the follow-the-regularized-leader (FTRL)-Proximal framework (McMahan, 2011). The updating rule of the FTRL-Proximal is formulated as follows: $w_{t+1} = \arg\min_w \left(g_{1:t} w + t\Psi(w) + \frac{1}{2} \sum_{s=1}^{t} \|Q_s^{\frac{1}{2}} (w - w_s)\|_2^2 \right)$, where $g_{1:t} w$ approximates $\ell_{1:t}$ based on the gradients $g_t = \nabla \ell_t(w_t)$, and the matrices Q_s represent generalized learning rates.

4. Neuronal Dynamics-based Online Training

4.1. Observation from Neuronal Dynamics

In this section, we will review the *discrete temporal dependency representation* for SNNs, $\boldsymbol{\epsilon}[t]$, and introduce the *continuous temporal dependency representation* for neuronal dynamics, $\mathbf{e}[t]$. The observation that the two representations are closely related motivates the proposed method discussed in Sect. 4.2.

The LIF model captures the biological neuron functioning through three fundamental processes: *charging*, *leakage*, and *firing*. These processes are vital for understanding the temporal dependencies within the neuronal (membrane) dynamics. When examining the temporal information flow from $\mathbf{u}(t)$ to $\mathbf{u}(t+1)$, we can observe three distinct components: *charging* from presynaptic inputs $\mathbf{s}(t) \nrightarrow \mathbf{u}(t+1)$, direct *leakage* or decaying $\mathbf{u}(t) \to \mathbf{u}(t+1)$, and the *firing* procedure mediated by spikes $\mathbf{u}(t) \nrightarrow \mathbf{s}(t)$.

***Discrete* temporal dependency representation in SNNs.** As shown in Fig. 1, BPTT has to maintain the computational graph of the previous time-steps and backpropagate through time. In SNNs, the temporal dependency lies in the dynamics of each spiking neuron, i.e. $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{u}^l[i]}$ and $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]}$ in Eq. (5). We further define

$$\boldsymbol{\epsilon}^l[t] = \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \tag{6}$$

as the *discrete* temporal dependency/sensitivity of $\mathbf{u}^l[t+1]$ with respect to $\mathbf{u}^l[t]$, represented by the colored arrows in Fig. 1. Then the BPTT in Eq. (5) can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \prod_{i=k}^{t-1} \boldsymbol{\epsilon}^l[i] \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)}_{\text{temporal component}} . \tag{7}$$

If we apply surrogate derivatives to $\frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]}$, then we have surrogate gradient methods (Chowdhury et al., 2022; Rathi & Roy, 2021; Rathi et al., 2019). If we do not apply surrogate derivatives and instead set $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]} \approx 0$, then OTTT (Xiao et al., 2022) is obtained.

An Overview of OTTT: With $\prod_{i=k}^{t-1} \boldsymbol{\epsilon}^l[i] = \lambda^{t-k}$, the gradient calculation in OTTT is degenerated to

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left(\sum_{k \leqslant t} \lambda^{t-k} \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right) .$$

***Continuous* temporal dependency representation for neuronal dynamics.** We encapsulate the entire temporal dependency from $\mathbf{u}(t)$ to $\mathbf{u}(t+1)$ using the notation $\mathbf{u}(t) \rightsquigarrow \mathbf{u}(t+1)$. This concise representation is expressed through the implicit function $\mathbf{u}(t+1) = I_m(\mathbf{u}(t))$, where $I_m(\cdot)$ effectively captures the complex interactions of *charging*, *leakage*, and *firing* dynamics in the LIF model.

To derive the derivatives of membrane potential $\mathbf{u}(t)$ using the implicit function $\mathbf{u}(t) \rightsquigarrow \mathbf{u}(t+1)$, we utilize the chain rule. First, we express the derivative $\frac{d\mathbf{u}(t+1)}{dt}$ and apply the chain rule as follows: $\frac{d\mathbf{u}(t+1)}{dt} = \frac{\partial I_m}{\partial \mathbf{u}(t)} \frac{\partial \mathbf{u}(t)}{\partial t} = \frac{\partial \mathbf{u}(t+1)}{\partial \mathbf{u}(t)} \frac{\partial \mathbf{u}(t)}{\partial t}$. This leads to the definition of $\mathbf{e}(t)$, representing the *continuous* temporal dependency, expressed as:

$$\mathbf{e}(t) \triangleq \frac{\partial \mathbf{u}(t+1)}{\partial \mathbf{u}(t)} = \mathbf{u}'(t+1) \oslash \mathbf{u}'(t)$$

Here, $\mathbf{e}(t)$ encapsulates the *continuous* temporal dependency from $\mathbf{u}(t)$ to $\mathbf{u}(t+1)$. The symbol $\oslash$ (or $/$) denotes element-wise division.

Combining Eq. (1), we have $\mathbf{e}(t) = \frac{\mathbf{u}(t+1) - \mathbf{I}(t+1)}{\mathbf{u}(t) - \mathbf{I}(t)}$. This representation illuminates the intricate temporal relationships within the neuronal dynamics and holds true for any continuous time t. When evaluating this at discrete time-steps $[t]$ across different layers in SNNs, we get $\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t+1] - \mathbf{I}^l[t+1]}{\mathbf{u}^l[t] - \mathbf{I}^l[t]}$. Combining Eq. (4), the *continuous* temporal dependency representation $\mathbf{e}^l[t]$ becomes [2]

$$\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t] - V_{th} \mathbf{s}^l[t]}{\mathbf{u}^l[t-1] - V_{th} \mathbf{s}^l[t-1]} . \tag{8}$$

[2] When $t = 1$, $\mathbf{e}^l[t] = \mathbf{u}^l[t] - V_{th} \mathbf{s}^l[t]$. Refer to Appendix A.2 for "numerical stability enhancing strategy".

4.2. Derivation of NDOT

As discussed in Sect. 4.1, within the BPTT framework, $\boldsymbol{\epsilon}^l[t]$ encapsulates the *discrete* temporal dependency of $\mathbf{u}^l[t+1]$ with respect to $\mathbf{u}^l[t]$ in SNNs. Similarly, our analysis shows that $\mathbf{e}(t)$ captures the *continuous* temporal dependency of $\mathbf{u}(t+1)$ with respect to $\mathbf{u}(t)$ from the perspective of neuronal dynamics. Given that $\mathbf{e}[t]$ and $\boldsymbol{\epsilon}^l[t]$ represent the same characteristic, a natural question arises: why not employ *neuronal dynamics-based* $\mathbf{e}[t]$ for computing gradients in SNNs? This question motivates our new method, Neuronal Dynamics-based Online Training (NDOT).

Algorithm 1 One iteration of NDOT for training SNNs

Input: Training data $(\mathbf{x}, \mathbf{y})$, Time-steps T.
Output: Trained network parameters $\{\mathbf{W}^l\}_{l=1}^N$.
for $t = 1, 2, \cdots, T$ **do**
  for $l = 1, 2, \cdots, N$ *// Forward* **do**
    Update $\mathbf{u}^l[t]$ and generate spikes $\mathbf{s}^l[t]$;
    Update $\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]}{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]}$;
    Update $\hat{\mathbf{a}}^l[t] = \mathbf{e}^l[t-1] \odot \hat{\mathbf{a}}^l[t-1] + \mathbf{s}^l[t]$ as the *temporal gradients*;
  end for
  for $l = N, N-1, \cdots, 1$ *// Backward* **do**
    Calculate the *spatial gradients* $\mathbf{g}_{\mathbf{u}^l}[t]$ at t;
    Calculate the instantaneous gradients $\nabla_{\mathbf{W}^l}\mathcal{L} = \mathbf{g}_{\mathbf{u}^l}[t]\hat{\mathbf{a}}^{l-1}[t]^\top$;
    Update $\mathbf{W}^l$ with $\nabla_{\mathbf{W}^l}\mathcal{L}$ based on the gradient-based optimizer.
  end for
end for

To enable forward-in-time online learning, we decouple the ***full gradients*** into *temporal components* and *spatial components*, as illustrated in Fig. 1. We redefine the temporal component gradients[3] in Eq. (7) as follows:

$$\hat{a}^{l-1}[t] \triangleq \frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \prod_{i=k}^{t-1} \mathbf{e}^{l-1}[i] \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} . \quad (9)$$

We further define $\mathbf{P}^l_{k,t} \triangleq \prod_{i=k}^{t-1} \mathbf{e}^l[i] = \mathbf{e}^l[k] \odot \cdots \odot \mathbf{e}^l[t-1] = \frac{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]}{\mathbf{u}^l[k-1] - V_{th}\mathbf{s}^l[k-1]}$. Using this, we have the gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)$$

$$\nabla_{\mathbf{W}^l}\mathcal{L} = \sum_{t=1}^{T} \mathbf{g}_{\mathbf{u}^l}[t] \left(\mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] \right)^\top , \quad (10)$$

where $\mathbf{g}_{\mathbf{u}^l}[t] \triangleq \left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \right)^\top$ is the *spatial gradient* for $\mathbf{u}^l[t]$. Based on Eq. (10), we can track *the temporal gradient* (also called the presynaptic activities):

$$\hat{\mathbf{a}}^{l-1}[t] = \mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] . \quad (11)$$

In the forward procedure, the relationship between two successive time-steps can be formulated as follows:

$$\hat{\mathbf{a}}^{l-1}[t] = \mathbf{e}^{l-1}[t-1] \odot \hat{\mathbf{a}}^{l-1}[t-1] + \mathbf{s}^{l-1}[t] . \quad (12)$$

The pseudo-code is provided in Algorithm 1. More details can be found in Appendix A.1.

Remark 4.1. At each time-step, given $\mathbf{g}_{\mathbf{u}^l}[t]$, the full gradients can be calculated independently by $\nabla_{\mathbf{W}^l}\mathcal{L} = \mathbf{g}_{\mathbf{u}^l}[t]\hat{\mathbf{a}}^{l-1}[t]^\top$ without backpropagation through time over $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{u}^l[i]}$ and $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]}$.

Remark 4.2. In the full gradient $\nabla_{\mathbf{W}^l}\mathcal{L} = \mathbf{g}_{\mathbf{u}^l}[t]\hat{\mathbf{a}}^{l-1}[t]^\top$, $\mathbf{g}_{\mathbf{u}^l}[t]$ represents the ***spatial gradient*** along *the spatial dimension*, while $\hat{\mathbf{a}}^{l-1}[t]$ denotes the ***temporal gradient*** along *the temporal dimension*. This decoupled gradient calculation enables forward-in-time learning.

So the remaining question is how to compute $\mathbf{g}_{\mathbf{u}^l}[t]$. We will address this in the following.

Instantaneous Loss. In typical SNN training, the classification is usually based on the average firing rate, $\frac{1}{T}\sum_{t=1}^{T} \mathbf{s}^N[t]$. The *offline loss function* $\mathcal{L}_{off}(\cdot)$ depends on $\{\mathbf{s}^N[1], \cdots, \mathbf{s}^N[T]\}$ at all time-steps and does not support online learning. To facilitate online learning, we use the instantaneous loss $\mathcal{L}[t]$. The two types of losses are defined as follows: $\mathcal{L}_{off} = \ell(\frac{1}{T}\sum_{t=1}^{T} \mathbf{s}^N[t], \mathbf{y})$, $\mathcal{L}[t] = \frac{1}{T}\ell(\mathbf{s}^N[t], \mathbf{y})$, where $\mathbf{y}$ is the label, $\mathbf{s}^N[t]$ are the output spike trains at the last layer, and $\ell(\cdot)$ can be the cross-entropy (CE) loss. The total loss $\mathcal{L} \triangleq \sum_{t=1}^{T} \mathcal{L}[t]$ is the upper bound of $\mathcal{L}_{off}$ when $\ell(\cdot)$ is a convex function.

Previous studies such as OTTT (Xiao et al., 2022) and TET (Deng et al., 2021) have shown that incorporating the mean-square-error (MSE) loss for training SNNs can enhance the accuracy. Building upon this, we combine CE loss and MSE loss, using it as the ***NDOT loss***, i.e.

$$\mathcal{L}[t] = (1-\alpha)\cdot\ell_{CE}(\mathbf{s}^N[t], \mathbf{y}) + \alpha\cdot\ell_{MSE}(\mathbf{s}^N[t], \mathbf{y}) . \quad (13)$$

With the instantaneous loss, the gradient for $\mathbf{u}^l[t]$, i.e. $\mathbf{g}_{\mathbf{u}^l}[t]$, can be independently calculated at each time-step without information from other time-steps,

$$\mathbf{g}_{\mathbf{u}^l}[t] = \left(\frac{\partial \mathcal{L}[t]}{\partial \mathbf{s}^N[t]} \prod_{j=l}^{N-1} \frac{\partial \mathbf{s}^{j+1}[t]}{\partial \mathbf{u}^j[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \right)^\top . \quad (14)$$

The *spatial gradients* $\mathbf{g}_{\mathbf{u}^l}[t]$ represent gradients between layers and can be efficiently computed using PyTorch's auto-grad functionality.

[3] We use $\odot$ for element-wise multiplication.

Our NDOT method is proposed to tackle the challenge of excessive memory consumption inherent in traditional approaches like BPTT by mitigating the need for storing vast amounts of intermediate information at all time-steps. Our proposed NDOT method enables forward-in-time online training of SNNs by using a "Continuous temporal dependency/sensitivity representation for neuronal dynamics" rather than using the "discrete temporal dependency/sensitivity in SNNs", hereby accurately capturing the temporal dependencies. Central to our method is the decomposition of the full gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$ into spatial gradient $g_{u^l}[t]$ and temporal gradient $\hat{a}^{l-1}[t]$, as shown in Fig. 1 and detailed in Algorithm 1. The spatial gradients $g_{u^l}[t]$ are efficiently computed with the PyTorch's auto-grad functionality, while the temporal gradients $\hat{a}^{l-1}[t]$ are recursively calculated in a "forward" manner, as described in Eq. (12).

4.3. Intuition Behind the NDOT Method

In this section, we will demonstrate that our NDOT method effectively utilizes temporal dependencies, even more so than methods that explicitly use temporal dependencies.

To understand how the NDOT method *implicitly* leverages temporal dependencies, we draw inspiration from the Follow-the-Regularized-Leader (FTRL) algorithm. We compare the results of FTRL-OTTT, which *explicitly* uses historical temporal information, with our NDOT method, which *implicitly* captures temporal dependencies through neuronal dynamics. Many state-of-the-art (SOTA) online SNN training algorithms, such as those discussed by Xiao et al. (2022), rely on instantaneous loss, potentially limiting their ability to exploit temporal dependencies. In contrast, our NDOT method also employs instantaneous loss but maximizes the use of temporal dependencies by incorporating neural dynamics in the gradient calculations.

FTRL-OTTT: Explicit utilization of historical information. Inspired by FTRL's explicit leveraging historical information, we introduce FTRL-OTTT, a two-phase learning strategy designed to address the limitations of instantaneous loss in OTTT (Xiao et al., 2022). In the first phase, we train an SNN with a small simulation time-step (e.g. $T = 2$) using extended epochs to obtain optimal solutions/weights, denoted as $\hat{\mathbf{W}}$. In the second phase, we use a larger target time-step (e.g. $T = 4$) and formulate an optimization problem that balances minimizing the instantaneous loss while remaining close to the optimal weights obtained in the first phase. The ***FTRL-OTTT loss*** is defined as

$$\hat{\mathcal{L}}[t] = \mathcal{L}[t] + \rho\|\mathbf{W} - \hat{\mathbf{W}}\|_2^2 \text{ or } \rho\|\mathbf{W} - \hat{\mathbf{W}}\|_1 \,.$$

Here, $\hat{\mathbf{W}}$ represents the optimal weights, summarizing the historical information and **explicitly** included as an FTRL regularization term. This two-stage FTRL-OTTT approach combines instantaneous and historical information to fully exploit temporal dependencies, enhancing the training efficiency and overall performance of existing SOTA online SNN training algorithms.

A related approach, Time Inheritance Training (TIT) in TET (Deng et al., 2021), addresses the training time issue by initializing weights for larger simulation time-steps with optimal weights $\hat{\mathbf{W}}$ obtained from smaller simulation time-steps. However, TET (Deng et al., 2021) only uses $\hat{\mathbf{W}}$ as initialization values and continues to optimize the original loss function.

NDOT Method: Implicit utilization of temporal dependencies. Although our NDOT method does not explicitly include an FTRL regularization term in its total loss, it works similarly by implicitly capturing temporal dependencies through neuronal dynamics. This is analogous to how FTRL's use of historical information stored in $\hat{\boldsymbol{W}}$, which helps us understand the intuition behind the NDOT method.

Experimental observations of FTRL-OTTT in Table 1 show substantial performance improvements when the FTRL loss is incorporated into other online training algorithms, such as OTTT (Xiao et al., 2022). This inclusion clarifies the intuition behind our proposed NDOT method. In summary, the NDOT method captures temporal dependencies through neuronal dynamics, functioning similarly to the explicit use of FTRL regularization in capturing historical information.

5. Experiments

In this section, we conduct extensive experiments on CIFAR-10 (Krizhevsky et al., 2009), CIFAR100 (Krizhevsky et al., 2009), and CIFAR10-DVS (Li et al., 2017) to demonstrate the superior performance of our proposed NDOT method on large-scale static and neuromorphic datasets. In our experiments, we use the VGG network architecture (64C3-128C3-AP2-256C3-256C3-AP2-512C3-512C3-AP2-512C3-512C3-GAP-FC). We use the SGD optimizer with no weight decay. The initial learning rate is 0.1 and will cosine decay to 0 during the training for all experiments. For the hyperparameters of LIF neuron models, we set $V_{th} = 1, \lambda = 0.5$. For our proposed NDOT method, we offer two parameter update strategies. NDOT$_O$: this variant represents the online real-time update, where parameters are updated immediately at each time-step before proceeding to the next calculation. NDOT$_A$: this variant represents the accumulated-time update, where gradients are accumulated over T time-steps before performing parameter updates. For detailed training procedures, please refer to Appendix A.3.

In practice, most BPTT with SG methods leverage batch normalization (BN) along the temporal dimension to achieve high performance with extremely low latency (Zheng et al., 2021; Duan et al., 2022), which requires calculating the mean and variance statistics for all time steps during the

Table 1. Comparison between the proposed NDOT method and other SOTA methods in SNNs.

Dataset	Model	Method	Arch.	Time-steps	Acc. (%)
CIFAR-10	SPIKE-NORM (Sengupta et al., 2019)	ANN-to-SNN	VGG-16	2500	91.55
	ReLuTS (Deng & Gu, 2020)	ANN-to-SNN	VGG-16	16	92.29
	QCFS (Bu et al., 2021)	ANN-to-SNN	VGG-16	4	93.96
	SlipReLu (Jiang et al., 2023)	ANN-to-SNN	ResNet-18	1	93.11
	BNTT (Kim & Panda, 2021)	BPTT	VGG-9	20	90.30
	TEBN (Duan et al., 2022)	BPTT	VGG-9	4	92.81
	tdBN (Zheng et al., 2021)	BPTT	ResNet-19	4	92.92
	SLTT (Meng et al., 2023)	BPTT	ResNet-18	6	94.59
	LTL (Yang et al., 2022)	Tandem Learning	VGG-11	16	93.20
	OTTT (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	6	93.73
	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	6	**94.89**
				4	**94.79**
				2	**94.44**
				1	**94.28**
	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	6	**94.90**
				4	**94.86**
				2	**94.41**
CIFAR-100	SPIKE-NORM (Sengupta et al., 2019)	ANN-to-SNN	VGG-16	2500	70.90
	SlipReLu (Jiang et al., 2023)	ANN-to-SNN	ResNet-18	4	74.89
	Hybrid (Rathi et al., 2019)	Hybrid	VGG-11	125	67.87
	BNTT (Kim & Panda, 2021)	BPTT	VGG-11	50	66.60
	TEBN (Duan et al., 2022)	BPTT	VGG-11	4	74.37
	TET (Deng et al., 2021)	BPTT	ResNet-19	4	74.62
	LTL (Yang et al., 2022)	Tandem Learning	VGG-11	16	72.63
	OTTT (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	6	71.11
	FTRL-OTTT	Forward-in-time	VGG-11 (WS)	6	75.89
				4	74.95
				2	74.55
	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	6	**76.61**
				4	**76.18**
				2	**75.27**
				1	**73.24**
	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	6	**76.47**
				4	**76.12**
				2	**75.01**
DVS-CIFAR10	NeuNorm (Wu et al., 2019)	BPTT	7-layer CNN	40	60.50
	BNTT (Kim & Panda, 2021)	BPTT	7-layer CNN	20	63.20
	tdBN (Zheng et al., 2021)	BPTT	ResNet-19	10	67.80
	TEBN (Duan et al., 2022)	BPTT	7-layer CNN	10	75.10
	PLIF (Fang et al., 2021b)	BPTT	7-layer CNN	20	74.80
	SLTT (Meng et al., 2023)	BPTT	VGG-11	10	77.30
	OTTT (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	10	77.10
	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	10	**77.50**
	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	10	**77.40**

forward procedure, making it incompatible with online gradients. To overcome this, we opt for a normalization-free approach inspired by ResNets (NF-ResNets) (Brock et al., 2020; 2021) and replace BN with scaled weight standardization (WS). WS standardizes weights by $\hat{\mathbf{W}}_{i,j} = \gamma \frac{\mathbf{W}_{i,j} - \mu_{W_i,}}{\sigma_{W_i,}\sqrt{N}}$, where γ is a scale parameter.

5.1. Comparison of Performance

We compare our NDOT approach with other SOTA work for SNN training, including ANN-to-SNN conversion (Bu et al., 2021; Jiang et al., 2023), BPTT with SG (Duan et al., 2022; Deng et al., 2021), and other online learning methods (Xiao et al., 2022). Among them, OTTT (Xiao et al., 2022) enables direct forward-in-time training on large-scale datasets with relatively low training costs, yet when tracking the temporal gradients, OTTT uses an approximation by setting $\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]} \approx 0$ and arguing that the derivative of the Heaviside step function is 0 almost everywhere. While in our NDOT, we use $\mathbf{e}^{l-1}[t]$ to capture the temporal dependency and leverage the temporal gradients $\hat{\mathbf{a}}^{l-1}[t]$ to facilitate forward-in-time learning.

We compare our proposed NDOT method with OTTT under the same experimental settings of network structures and total time-steps. In order to understand better the effectiveness of our proposed NDOT method, we employ OTTT's variant with FTRL loss, termed as FTRL-OTTT. The results are shown in Table 1. NDOT outperforms OTTT on all the datasets regarding accuracy under the same time-step, indicating the superior efficiency of NDOT.

On CIFAR-10, our NDOT method outperforms all existing approaches, achieving the highest accuracy. Even with $T = 1$, NDOT shows a 1.08% improvement compared to LTL (93.20%) with a simulation length of $T = 16$. The superiority of NDOT is more evident on CIFAR-100, with an accuracy increase exceeding 5% on VGG compared to OTTT at $T = 6$. Notably, NDOT's advantage is more pronounced on complex datasets like CIFAR-100. We get comparable results from FTRL-OTTT and NDOT, indicating that explicitly leveraging historical information enhances network performance, highlighting the effectiveness of NDOT in capturing precise temporal dependencies. For more experimental results, please refer to Appendix A.4.

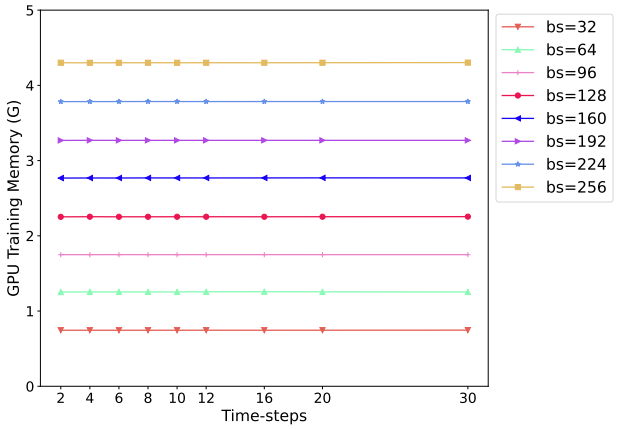

Figure 2. Training memory costs of NDOT with different batch size under different time-steps.

5.2. The Training Memory Costs

A major advantage of our proposed NDOT method is that it does not require backpropagation along the temporal dimension. This means NDOT maintains only constant training memory costs regardless of the number of time-steps used, which avoids the large memory costs of BPTT.

We verify this by training the VGG network on CIFAR-100 with batch sizes ranging from 32 to 256 across different time-steps, then calculating the memory costs on the GPU. As shown in Fig. 2, the training memory of NDOT remains constant for a given batch size, regardless of the number of time-steps. Indeed, this advantage also allows for potential training acceleration of SNNs by using larger batch sizes with the same computational resources.

5.3. Ablation Study on Hyper-parameter α in Loss $\mathcal{L}[t]$

To comprehensively examine the impact of the hyper-parameter α in Eq. (13), we conducted experiments on DVS-CIFAR10 with a fixed time-step of $T = 6$ in the online setting, varying α from 0.0 to 1.0 in increments of 0.1. The results are summarized in Table 2.

Table 2. Effect of Different α on DVS-CIFAR10 with Time-step $T = 6$ using our NDOT$_O$ method.

α	0.0	0.05	**0.1**	0.2	0.3	0.4
Accuracy (%)	74.2	75.1	**75.6**	75.1	74.8	74.7
α	0.5	0.6	0.7	0.8	0.9	1.0
Accuracy (%)	73.8	74.1	73.1	73.7	72.6	71.0

From Table 2, it is evident that varying α leads to fluctuations in model performance. (a) Notably, an α value of 0.1 yields the highest accuracy of 75.6%, indicating optimal performance under these experimental conditions. However, as α deviates further from this value, the accuracy tends to decline gradually. For instance, at $\alpha = 0.0$ and $\alpha = 1.0$, the accuracy decreases significantly to 74.2% and 71.0%, respectively. (b) This analysis suggests that a balanced combination of cross-entropy and mean-square-error losses, represented by an intermediate α value, contributes to optimal model performance.

6. Conclusion

This paper addresses the forward-in-time training challenge in SNNs, aiming to avoid unfolding the computation graph along the temporal dimension and reduce memory costs. We propose the Neuronal Dynamics-based Online Training (NDOT) method, which incorporates neuronal dynamics into the gradient calculation. By combining spatial gradients across layers with temporal gradients, our NDOT method enables full gradient representation and facilitates forward-in-time training. Extensive experiments demonstrate the consistent superiority of our DNOT method. The FTRL algorithm helps illustrate the intuition behind NDOT's use of the temporal dependencies. FTRL explicitly utilizes historical information, similar to how NDOT captures temporal dependencies through neuronal dynamics, as confirmed by experimental results.

Acknowledgements

This work is part of the research project ("Energy-based probing for Spiking Neural Networks", Contract No. TII/ARRC/2073/2021) in collaboration between Technology Innovation Institute (TII, Abu Dhabi) and Mohamedbin Zayed University of Artificial Intelligence (MBZUAI, Abu Dhabi).

Impact Statement

Our research aims to advance the field of Machine Learning, specifically focusing on energy-efficient AI algorithms. While there are many potential societal impacts of our work, none require specific highlighting at this time.

Our research study primarily focuses on the direct training of high-performance and low-latency SNNs, which does not bring obvious negative effects. Additionally, due to their energy-saving properties, SNNs are likely to become essential in edge computing applications as their adoption increases.

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.-J., 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.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., and Maass, W. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

Bohnstingl, T., Woźniak, S., Pantazi, A., and Eleftheriou, E. Online spatio-temporal learning in deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Bottou, L. and Bousquet, O. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

Brock, A., De, S., and Smith, S. L. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *International Conference on Learning Representations*, 2020.

Brock, A., De, S., Smith, S. L., and Simonyan, K. High-performance large-scale image recognition without normalization. In *International Conference on Machine Learning*, pp. 1059–1071. PMLR, 2021.

Bu, T., Fang, W., Ding, J., DAI, P., Yu, Z., and Huang, T. Optimal ann-snn conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*, 2021.

Chowdhury, S. S., Rathi, N., and Roy, K. Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning. In *European Conference on Computer Vision*, pp. 709–726. Springer, 2022.

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

Deng, L., Wu, Y., Hu, X., Liang, L., Ding, Y., Li, G., Zhao, G., Li, P., and Xie, Y. Rethinking the performance comparison between snns and anns. *Neural networks*, 121: 294–307, 2020.

Deng, S. and Gu, S. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2020.

Deng, S., Li, Y., Zhang, S., and Gu, S. Temporal efficient training of spiking neural network via gradient reweighting. In *International Conference on Learning Representations*, 2021.

Deng, S., Lin, H., Li, Y., and Gu, S. Surrogate module learning: Reduce the gradient error accumulation in training spiking neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., and Pfeiffer, M. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)*, pp. 1–8. ieee, 2015.

Duan, C., Ding, J., Chen, S., Yu, Z., and Huang, T. Temporal effective batch normalization in spiking neural networks. *Advances in Neural Information Processing Systems*, 35: 34377–34390, 2022.

Duchi, J. and Singer, Y. Efficient online and batch learning using forward backward splitting. *The Journal of Machine Learning Research*, 10:2899–2934, 2009.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Duchi, J. C., Shalev-Shwartz, S., Singer, Y., and Tewari, A. Composite objective mirror descent. In *COLT*, volume 10, pp. 14–26. Citeseer, 2010.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., and Tian, Y. Deep residual learning in spiking neural networks. *Advances in Neural Information Processing Systems*, 34:21056–21069, 2021a.

Fang, W., Yu, Z., Chen, Y., Masquelier, T., Huang, T., and Tian, Y. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2661–2671, 2021b.

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

Guo, Y., Chen, Y., Zhang, L., Liu, X., Wang, Y., Huang, X., and Ma, Z. Im-loss: information maximization loss for spiking neural networks. *Advances in Neural Information Processing Systems*, 35:156–166, 2022.

Hazan, E., Rakhlin, A., and Bartlett, P. Adaptive online gradient descent. *Advances in neural information processing systems*, 20, 2007.

Huh, D. and Sejnowski, T. J. Gradient descent for spiking neural networks. *Advances in neural information processing systems*, 31, 2018.

Jiang, H., Anumasa, S., De Masi, G., Xiong, H., and Gu, B. A unified optimization framework of ann-snn conversion: Towards optimal mapping from activation values to firing rates. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Kag, A. and Saligrama, V. Training recurrent neural networks via forward propagation through time. In *International Conference on Machine Learning*, pp. 5189–5200. PMLR, 2021.

Kaiser, J., Mostafa, H., and Neftci, E. Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *Frontiers in Neuroscience*, 14:424, 2020.

Kiefer, J. and Wolfowitz, J. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

Kim, Y. and Panda, P. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in neuroscience*, pp. 1638, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Langford, J., Li, L., and Zhang, T. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10(3), 2009.

Lee, J. H., Delbruck, T., and Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

Li, H., Liu, H., Ji, X., Li, G., and Shi, L. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017.

Li, Y., Deng, S., Dong, X., Gong, R., and Gu, S. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *International conference on machine learning*, pp. 6316–6325. PMLR, 2021a.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., and Gu, S. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34:23426–23439, 2021b.

Li, Y., Kim, Y., Park, H., Geller, T., and Panda, P. Neuromorphic data augmentation for training spiking neural networks. In *European Conference on Computer Vision*, pp. 631–649. Springer, 2022.

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

McMahan, B. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 525–533. JMLR Workshop and Conference Proceedings, 2011.

McMahan, H. B. and Streeter, M. Adaptive bound optimization for online convex optimization. *COLT 2010*, pp. 244, 2010.

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., and Luo, Z.-Q. Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12444–12453, 2022.

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., and Luo, Z.-Q. Towards memory-and time-efficient backpropagation for training spiking neural networks. *arXiv preprint arXiv:2302.14311*, 2023.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., and Graves, A. Practical real time recurrent learning with a sparse approximation. In *International conference on learning representations*, 2020.

Mujika, A., Meier, F., and Steger, A. Approximating real-time recurrent learning with random kronecker factors. *Advances in Neural Information Processing Systems*, 31, 2018.

Neftci, E. O., Mostafa, H., and Zenke, F. 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.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., Wang, G., Zou, Z., Wu, Z., He, W., et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019.

Rathi, N. and Roy, K. 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.

Rathi, N., Srinivasan, G., Panda, P., and Roy, K. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International Conference on Learning Representations*, 2019.

Robbins, H. and Monro, S. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Roy, K., Jaiswal, A., and Panda, P. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pp. 807–814, 2007.

Shalev-Shwartz, S. et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Shrestha, S. B. and Orchard, G. Slayer: Spike layer error reassignment in time. *Advances in Neural Information Processing Systems*, 31, 2018.

Tallec, C. and Ollivier, Y. Unbiased online recurrent optimization. In *International Conference on Learning Representations*, 2018.

Thiele, J. C., Bichler, O., and Dupret, A. Spikegrad: An ann-equivalent computation model for implementing backpropagation with spikes. In *International Conference on Learning Representations*, 2019.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Wu, H., Zhang, Y., Weng, W., Zhang, Y., Xiong, Z., Zha, Z.-J., Sun, X., and Wu, F. Training spiking neural networks with accumulated spiking flow. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10320–10328, 2021.

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

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

Xiao, L. Dual averaging method for regularized stochastic learning and online optimization. *Advances in Neural Information Processing Systems*, 22, 2009.

Xiao, M., Meng, Q., Zhang, Z., Wang, Y., and Lin, Z. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. *Advances in Neural Information Processing Systems*, 34:14516–14528, 2021.

Xiao, M., Meng, Q., Zhang, Z., He, D., and Lin, Z. Online training through time for spiking neural networks. *Advances in Neural Information Processing Systems*, 35: 20717–20730, 2022.

Yang, Q., Wu, J., Zhang, M., Chua, Y., Wang, X., and Li, H. Training spiking neural networks with local tandem learning. *Advances in Neural Information Processing Systems*, 35:12662–12676, 2022.

Yin, B., Corradi, F., and Bohté, S. M. Accurate online training of dynamical spiking neural networks through forward propagation through time. *Nature Machine Intelligence*, pp. 1–10, 2023.

Zenke, F. and Ganguli, S. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

Zenke, F. and Vogels, T. P. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4): 899–925, 2021.

Zheng, H., Wu, Y., Deng, L., Hu, Y., and Li, G. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11062–11070, 2021.

Zhou, S., Li, X., Chen, Y., Chandrasekaran, S. T., and Sanyal, A. Temporal-coded deep spiking neural network with easy training and robust performance. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11143–11151, 2021.

A.1. Detailed Derivation of NDOT

A.1.1. Derivation of Eq. (10)

The gradients of BPTT with T time-steps are calculated by

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left\{ \frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \prod_{i=k}^{t-1} \left(\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{u}^l[i]} + \frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]} \right) \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right\}}_{\text{temporal component}}$$

We further define

$$\boldsymbol{\epsilon}^l[t] = \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}$$

as the neuronal temporal dependency/sensitivity of $\mathbf{u}^l[t+1]$ with respect to $\mathbf{u}^l[t]$ in the discrete condition, represented by the colored arrows in Fig. 1. Then BPTT in Eq. (5) can be rewritten as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \prod_{i=k}^{t-1} \boldsymbol{\epsilon}^l[i] \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)}_{\text{temporal component}}$$

All the temporal information going from $\mathbf{u}(t)$ to $\mathbf{u}(t+1)$ contains two parts: directly decaying $\mathbf{u}(t) \to \mathbf{u}(t+1)$ and two-step going through spikes $\mathbf{u}(t) \nrightarrow \mathbf{s}(t) \nrightarrow \mathbf{u}(t+1)$. We wrap all the temporal information going from $\mathbf{u}(t)$ to $\mathbf{u}(t+1)$ as $\mathbf{u}(t) \rightsquigarrow \mathbf{u}(t+1)$ and denote with an implicit function

$$\mathbf{u}(t+1) = I_m(\mathbf{u}(t)) \, .$$

Then the derivatives of membrane potential $\mathbf{u}(t)$ with respect to time t can be calculated through the implicit function, $\mathbf{u}(t) \rightsquigarrow \mathbf{u}(t+1)$, with the chain rule,

$$\frac{d\mathbf{u}(t+1)}{dt} = \frac{\partial I_m}{\partial \mathbf{u}(t)} \frac{\partial \mathbf{u}(t)}{\partial t} = \frac{\partial \mathbf{u}(t+1)}{\partial \mathbf{u}(t)} \frac{\partial \mathbf{u}(t)}{\partial t} \, .$$

This leads to the definition of $\boldsymbol{e}(t)$ as the neuronal temporal dependency or sensitivity, expressed as:

$$\mathbf{e}(t) \triangleq \frac{\partial \mathbf{u}(t+1)}{\partial \mathbf{u}(t)} = \frac{d\mathbf{u}(t+1)}{dt} \oslash \frac{d\mathbf{u}(t)}{dt} = \mathbf{u}'(t+1) \oslash \mathbf{u}'(t) = \frac{\mathbf{u}'(t+1)}{\mathbf{u}'(t)} \, .$$

Here, $\mathbf{e}(t)$ encapsulates the *continuous* neuronal temporal dependency or sensitivity of $\mathbf{u}(t+1)$ with respect to $\mathbf{u}(t)$, and the symbol $\oslash$ (or $/$) is element-wise division.

To summarize, for the continuous temporal dependency, we have

$$\mathbf{e}(t) = \frac{\partial \mathbf{u}(t+1)}{\partial \mathbf{u}(t)} = \frac{\mathbf{u}'(t+1)}{\mathbf{u}'(t)} \quad \text{(This means } \oslash \text{)}$$

$$\mathbf{e}(t) = \frac{\mathbf{u}(t+1) - \mathbf{I}(t+1)}{\mathbf{u}(t) - \mathbf{I}(t)} \quad \text{(Neuron Dynamics)} \, .$$

This representation illuminates the intricate temporal relationships within the neuronal dynamics, holding true for any continuous value of time t.

Evaluating this at discrete time-steps $[t]$ across different layers in SNNs, we obtain:

$$\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t+1] - \mathbf{I}^l[t+1]}{\mathbf{u}^l[t] - \mathbf{I}^l[t]} = \left(\mathbf{u}^l[t+1] - \mathbf{I}^l[t+1]\right) \oslash \left(\mathbf{u}^l[t] - \mathbf{I}^l[t]\right) \, .$$

Combining Eq. (4), we have

$$\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t+1] - \mathbf{I}^l[t+1]}{\mathbf{u}^l[t] - \mathbf{I}^l[t]} = \frac{\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]}{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]} = \left(\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]\right) \oslash \left(\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]\right) .$$

Here division represents the element-wise division.

As in SNNs, we use the discrete version of $\mathbf{u}^l[t]$ across different layers and consecutive time-steps, we can evaluate the continuous function $\mathbf{e}(t)$ at time-step t, as follows:

$$\begin{aligned}
\mathbf{e}^l[t] =& \frac{\mathbf{u}^l[t+1] - \mathbf{I}^l[t+1]}{\mathbf{u}^l[t] - \mathbf{I}^l[t]} \\
=& \frac{\mathbf{u}^l[t+1] - \mathbf{W}^l\mathbf{s}^{l-1}[t+1]}{\mathbf{u}^l[t] - \mathbf{W}^l\mathbf{s}^{l-1}[t]} \quad \text{(SNNs with Eq. (4))} \\
=& \frac{\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]}{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]} \quad \text{(Using Eq. (4) for } \mathbf{u}^l[t+1])
\end{aligned}$$

Then, we have

$$\mathbf{e}^l[t] = \frac{\mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t]}{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]} .$$

Here division represents the element-wise division.

By replacing the term $\boldsymbol{\epsilon}^l[t]$ in BPTT with $\mathbf{e}^{l-1}[t]$ and use the element-wise multiplication $\odot$ between the temporal components, we redefine the temporal component gradients in Eq. (7) as follows:

$$\hat{a}^{l-1}[t] \triangleq \frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t}\prod_{i=k}^{t-1} \mathbf{e}^{l-1}[i] \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} . \tag{S.1}$$

Remark A.1.1. We use element-wise multiplication $\odot$ to ensure the dimensions match in each layer without introducing additional dimensions. That is why we use $\mathbf{e}^{l-1}[i]$ in $\hat{a}^{l-1}[t]$.

Then we have

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left\{ \frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t}\prod_{i=k}^{t-1} \left(\frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{u}^l[i]} + \frac{\partial \mathbf{u}^l[i+1]}{\partial \mathbf{s}^l[i]} \frac{\partial \mathbf{s}^l[i]}{\partial \mathbf{u}^l[i]} \right) \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right\} \\
&\Longleftrightarrow \\
&\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t}\prod_{i=k}^{t-1} \boldsymbol{\epsilon}^l[i] \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)}_{\text{temporal component}} \\
&\quad \approx \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \mathbf{e}^{l-1}[t-1] \odot \mathbf{e}^{l-1}[t-2] \odot \cdots \odot \mathbf{e}^{l-1}[k+1] \odot \mathbf{e}^{l-1}[k] \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)}_{\text{temporal component}} \\
&\quad = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \underbrace{\left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t}\prod_{i=k}^{t-1} \mathbf{e}^{l-1}[i] \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)}_{\text{temporal component}} \\
&\Longrightarrow \\
&\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t}\prod_{i=k}^{t-1} \frac{\mathbf{u}^{l-1}[i] - V_{th}\mathbf{s}^{l-1}[i]}{\mathbf{u}^{l-1}[i-1] - V_{th}\mathbf{s}^{l-1}[i-1]} \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)
\end{aligned}$$

Further, we denote

$$\mathbf{P}^l_{k,t} \triangleq \prod_{i=k}^{t-1} \mathbf{e}^l[i] = \mathbf{e}^l[k] \odot \cdots \odot \mathbf{e}^l[t-1] = \frac{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]}{\mathbf{u}^l[k-1] - V_{th}\mathbf{s}^l[k-1]} .$$

Therefore, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \left(\frac{\partial \mathbf{u}^l[t]}{\partial \mathbf{W}^l} + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \frac{\partial \mathbf{u}^l[k]}{\partial \mathbf{W}^l} \right)$$

$$\nabla_{\mathbf{W}^l} \mathcal{L} = \sum_{t=1}^{T} \mathbf{g}_{\mathbf{u}^l}[t] \left(\mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] \right)^\top = \sum_{t=1}^{T} \mathbf{g}_{\mathbf{u}^l}[t] \left(\hat{a}^{l-1}[t] \right)^\top$$

where $\mathbf{g}_{\mathbf{u}^l}[t] = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \right)^\top$ is the gradient for $\mathbf{u}^l[t]$ along the spatial dimension. Based on Eq. (10), we can track presynaptic activities along the temporal/time dimension.

$$\hat{\mathbf{a}}^{l-1}[t] \triangleq \mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] .$$

For each neuron during the forward procedure, the relationship between two successive time-steps (from time-step $t-1$ to time-step t) can be formulated as

$$\hat{\mathbf{a}}^{l-1}[t] = \mathbf{e}^{l-1}[t-1] \odot \hat{\mathbf{a}}^{l-1}[t-1] + \mathbf{s}^{l-1}[t] ,$$

where

$$\mathbf{e}^{l-1}[t] = \frac{\mathbf{u}^{l-1}[t] - V_{th}\mathbf{s}^{l-1}[t]}{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]} = (\mathbf{u}^{l-1}[t] - V_{th}\mathbf{s}^{l-1}[t]) \oslash (\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]) .$$

A.1.2. Derivation from Eq. (9) to Eq. (12)

From the definition of $\mathbf{P}^l_{k,t}$, we have

$$\mathbf{P}^l_{k,t} \triangleq \prod_{i=k}^{t-1} \mathbf{e}^l[i] = \mathbf{e}^l[k] \odot \cdots \odot \mathbf{e}^l[t-1] = \frac{\mathbf{u}^l[t-1] - V_{th}\mathbf{s}^l[t-1]}{\mathbf{u}^l[k-1] - V_{th}\mathbf{s}^l[k-1]} .$$

Based on Eq. (9), we can get Eq. (10) as follows,

$$\begin{aligned}
\hat{\mathbf{a}}^{l-1}[t] &\triangleq \mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] \\
&= \mathbf{s}^{l-1}[t] + \sum_{k<t} \mathbf{e}^{l-1}[k] \odot \cdots \odot \mathbf{e}^{l-1}[t-1] \odot \mathbf{s}^{l-1}[k] \\
&= \mathbf{s}^{l-1}[t] + \sum_{k<t} \frac{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k]
\end{aligned}$$

Based on Eq. (10), we can track presynaptic activities along the temporal/time dimension at $t-1$,

$$\begin{aligned}
\hat{\mathbf{a}}^{l-1}[t-1] &\triangleq \mathbf{s}^{l-1}[t-1] + \sum_{k<t-1} \mathbf{e}^{l-1}[k] \odot \cdots \odot \mathbf{e}^{l-1}[t-2] \odot \mathbf{s}^{l-1}[k] \\
&= \mathbf{s}^{l-1}[t-1] + \sum_{k<t-1} \frac{\mathbf{u}^{l-1}[t-2] - V_{th}\mathbf{s}^{l-1}[t-2]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k]
\end{aligned}$$

Then

$$
\begin{aligned}
&\mathbf{e}^{l-1}[t-1] \odot \hat{\mathbf{a}}^{l-1}[t-1] \\
&= \mathbf{e}^{l-1}[t-1] \odot \left(\mathbf{s}^{l-1}[t-1] + \sum_{k<t-1} \frac{\mathbf{u}^{l-1}[t-2] - V_{th}\mathbf{s}^{l-1}[t-2]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k] \right) \\
&= \frac{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]}{\mathbf{u}^{l-1}[t-2] - V_{th}\mathbf{s}^{l-1}[t-2]} \odot \left(\mathbf{s}^{l-1}[t-1] + \sum_{k<t-1} \frac{\mathbf{u}^{l-1}[t-2] - V_{th}\mathbf{s}^{l-1}[t-2]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k] \right) \\
&= \frac{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]}{\mathbf{u}^{l-1}[t-2] - V_{th}\mathbf{s}^{l-1}[t-2]} \odot \mathbf{s}^{l-1}[t-1] + \sum_{k<t-1} \frac{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k] \\
&= \sum_{k<t} \frac{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]}{\mathbf{u}^{l-1}[k-1] - V_{th}\mathbf{s}^{l-1}[k-1]} \odot \mathbf{s}^{l-1}[k] \\
&= \sum_{k<t} \mathbf{P}^{l-1}_{k,t} \odot \mathbf{s}^{l-1}[k] \\
&= \hat{\mathbf{a}}^{l-1}[t] - \mathbf{s}^{l-1}[t]
\end{aligned}
$$

Therefore, we have the following, which is Eq. (12)

$$\hat{\mathbf{a}}^{l-1}[t] = \mathbf{e}^{l-1}[t-1] \odot \hat{\mathbf{a}}^{l-1}[t-1] + \mathbf{s}^{l-1}[t]$$

Similarly, for the l-th layer, we have

$$\hat{\mathbf{a}}^{l}[t] = \mathbf{e}^{l}[t-1] \odot \hat{\mathbf{a}}^{l}[t-1] + \mathbf{s}^{l}[t]$$

A.2. Numerical Stability Enhancing Strategy

To address potential numerical instabilities in the computations of $\mathbf{e}^{l-1}[t]$, i.e.,

$$\mathbf{e}^{l-1}[t] = \frac{\mathbf{u}^{l-1}[t] - V_{th}\mathbf{s}^{l-1}[t]}{\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]} = (\mathbf{u}^{l-1}[t] - V_{th}\mathbf{s}^{l-1}[t]) \oslash (\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1])$$

we implement a "numerical stability enhancing strategy" in our codes implementation. Specifically, when the denominator of $\mathbf{e}^{l-1}[t]$, $\mathbf{u}^{l-1}[t-1] - V_{th}\mathbf{s}^{l-1}[t-1]$, equals zero, we first determine the sign of $\mathbf{u}^{l-1}[t-1]$. Then we apply a clamping function to restrict the value within the range $[-\lambda, \lambda]$. These adjustments help mitigate numerical instability concerns and ensure the robustness of our computational approach.

A.3. Implementation Details

As introduced in Sect. 4.2, we will calculate instantaneous gradients $\frac{\partial \mathcal{L}[t]}{\partial \mathbf{W}^l} = \mathbf{g}_{\mathbf{u}^l}[t]\hat{a}^{l-1}[t]$ at each time step. We can choose to immediately update parameters before the calculation of the next time step, which we denote as NDOT$_O$, or we can accumulate the gradients by T time steps and then update parameters, which we denote as NDOT$_A$. For NDOT$_O$, we assume that the online update is small and has negligible effects for the following calculation.

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 (Duan et al., 2022; Li et al., 2021b), 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 (NF-ResNets) (Brock et al., 2020; 2021) and replace BN with scaled weight standardization (WS). SW standardizes weights by

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

where γ is a scale parameter.

A.3.1. Datasets

In this work, we conduct extensive experiments on CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009), and CIFAR10-DVS (Li et al., 2017) to demonstrate the superior performance of our proposed NDOT method on large-scale static and neuromorphic datasets.

CIFAR-10 CIFAR-10 consists of color images categorized into 10 classes of objects, comprising 50,000 training samples and 10,000 testing samples. Each sample is a $32 \times 32 \times 3$ color image. Our pre-processing involves normalizing the inputs based on global mean and standard deviation, along with employing random cropping, horizontal flipping, and cutout (DeVries & Taylor, 2017) for data augmentation. The inputs to the first layer of SNNs at each time step directly correspond to pixel values, akin to real-valued input currents.

CIFAR-100 CIFAR-100, akin to CIFAR-10, is a dataset featuring 100 classes of objects. With 50,000 training samples and 10,000 testing samples, it mirrors the structure of CIFAR-10. Our pre-processing methods align with those employed for CIFAR-10. Both CIFAR-10 and CIFAR-100 datasets are governed by the MIT License.

DVS-CIFAR10 The DVS-CIFAR10 dataset represents a neuromorphic adaptation of the CIFAR-10 dataset, generated by a Dynamic Vision Sensor (DVS). Comprising 10,000 samples, it constitutes one-sixth of the original CIFAR-10 dataset and is characterized by spike trains with two channels denoting ON-event and OFF-event spikes. The pixel dimension is expanded to 128×128. Following standard procedures, we partition the dataset into 9000 training samples and 1000 testing samples.

In terms of data pre-processing, we enhance efficiency by aggregating spike events into 2 time-steps and reducing the spatial resolution to 48×48 through interpolation. Similar to CIFAR-10 datasets, random cropping augmentation is applied to input data. Additionally, inputs are normalized using the global mean and standard deviation of all time steps, a process seamlessly integrated into the connection weights of the first layer

A.4. More Experiments

We have conducted additional experiments to evaluate the performance of our NDOT method on CIFAR-100 and DVS-CIFAR 10 datasets using both NDOT$_A$ and NDOT$_O$. The results are summarized in Table S1.

Based on the results, we have (a) For the CIFAR-100 dataset, our NDOT$_A$ method achieved an accuracy of 73.24% with time-step $T = 1$, surpassing OTTT$_A$ which attained an accuracy of 71.11% with a longer time-step $T = 6$. Notably, with time-step $T = 6$, our NDOT$_A$ method exhibited a substantial accuracy improvement of 5.35%, achieving an accuracy of 76.47%. (b) For the DVS-CIFAR10 dataset, both our NDOT$_A$ and NDOT$_O$ performed well even with a minimal time-step of 2. Specifically, Our NDOT$_A$ achieved an accuracy of 77.3% with time-step $T = 8$, surpassing OTTT$_A$'s accuracy of 76.30% with time-step $T = 10$. (c) These results highlight the superior performance of our NDOT methods, both NDOT$_O$ and NDOT$_A$, across different datasets (such as CIFAR-100 and DVS-CIFAR10 datasets) and time-steps, showcasing its robustness and versatility in training spiking neural networks.

Table S1. Results of our NDOT method (NDOT$_O$ and NDOT$_A$) on CIFAR-100 and DVS-CIFAR10 datasets with various time-steps.

Dataset	Model	Method	Architecture	Time-steps	Accuracy (%)
CIFAR-100	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	6	76.47
CIFAR-100	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	4	76.12
CIFAR-100	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	2	75.01
CIFAR-100	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	1	73.24
CIFAR-100	OTTT$_A$ (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	6	71.11
CIFAR-100	OTTT$_O$ (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	6	71.11
CIFAR-100	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	6	76.61
CIFAR-100	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	4	76.18
CIFAR-100	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	2	75.27
CIFAR-100	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	1	73.24
DVS-CIFAR10	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	10	77.4
DVS-CIFAR10	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	8	77.3
DVS-CIFAR10	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	6	76.0
DVS-CIFAR10	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	4	74.9
DVS-CIFAR10	NDOT$_A$ (Ours)	Forward-in-time	VGG-11 (WS)	2	71.1
DVS-CIFAR10	OTTT$_A$ (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	10	76.30
DVS-CIFAR10	OTTT$_O$ (Xiao et al., 2022)	Forward-in-time	VGG-11 (WS)	10	77.10
DVS-CIFAR10	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	10	77.5
DVS-CIFAR10	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	8	77.3
DVS-CIFAR10	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	6	75.6
DVS-CIFAR10	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	4	74.1
DVS-CIFAR10	NDOT$_O$ (Ours)	Forward-in-time	VGG-11 (WS)	2	71.1