Supplementary Materials for: Towards Memory- and Time-Efficient Backpropagation for Training Spiking Neural Networks

Qingyan Meng[1,2], Mingqing Xiao[3], Shen Yan[4], Yisen Wang[3,5], Zhouchen Lin[3,5,6,*], Zhi-Quan Luo[1,2]
[1]The Chinese University of Hong Kong, Shenzhen [2]Shenzhen Research Institute of Big Data
[3]Key Lab. of Machine Perception (MoE), School of Artificial Intelligence, Peking University
[4]Center for Data Science, Peking University [5]Institute for Artificial Intelligence, Peking University
[6]Peng Cheng Laboratory
qingyanmeng@link.cuhk.edu.cn, {mingqing_xiao, yanshen, yisen.wang, zlin}@pku.edu.cn, luozq@cuhk.edu.cn

A. Detailed Dynamics of Feedforward SNNs with LIF Neurons

An SNN operates by receiving an input time sequence and generating an output time sequence (binary spike train) through the iterative use of brain-inspired neuronal dynamics. We consider feedforward SNNs with LIF neurons in this work. Specifically, the network follows the difference equation:

$$\begin{cases} \mathbf{u}^l[t] = (1-\frac{1}{\tau})\mathbf{v}^l[t-1] + \mathbf{W}^l\mathbf{s}^{l-1}[t], \\ \mathbf{s}^l[t] = H(\mathbf{u}^l[t] - V_{th}), \\ \mathbf{v}^l[t] = \mathbf{u}^l[t] - V_{th}\mathbf{s}^l[t], \end{cases} \tag{S1}$$

where $l = 1, \cdots, L$ is the layer index, $t = 1, \cdots, T$ is the time step index, $\{\mathbf{s}^0[t]\}_{t=1}^T$ are the input to the network, and $\{\mathbf{s}^l[t]\}_{t=1}^T$ are the binary output sequence of the l-th layer, τ is a pre-defined time constant, $H(\cdot)$ is the the Heaviside step function, $\mathbf{u}^l[t]$ is the membrane potential before resetting, $\mathbf{v}^l[t]$ is the potential after resetting, and $\mathbf{W}^l$ are the weight to be trained. $\mathbf{v}^l[0]$ is set to be 0 for $l = 1, \cdots, L$.

The input $\mathbf{s}^0$ to the SNN can consist of either neuromorphic data or static data such as images. While neuromorphic data are inherently suitable for SNNs, when dealing with static data, a common approach is to repeatedly apply them to the first layer at each time step [31, 33, 26, 21]. This allows the first layer to serve as a spike-train data generator.

The output of the network $\{\mathbf{s}^L[t]\}_{t=1}^T$ is utilized for decision making. In classification tasks, a common approach involves computing $\mathbf{o} = \frac{1}{T}\sum_{t=1}^T \mathbf{W}^o\mathbf{s}^L[t]$, where $\mathbf{o} \in \mathbb{R}^c$ and c represents the number of classes. The input sequence $\{\mathbf{s}^0[t]\}_{t=1}^T$ is classified as belonging to the i-th class if $\mathbf{o}_i$ is the largest value among $\{\mathbf{o}_1, \cdots, \mathbf{o}_c\}$.

Due to the binary nature of spike communication between neurons, SNNs can be efficiently implemented on neuromorphic chips, enabling energy-efficient applications.

B. Derivation for Eqs. (9) and (10)

Recall that

$$\epsilon^l[t] \triangleq \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}, \tag{S2}$$

which is the dependency between $\mathbf{u}^l[t+1]$ and $\mathbf{u}^l[t]$. We derive Eq. (10) as below. We omit the derivation for Eq. (9) since it is a simple corollary of Eq. (10).

Lemma 1 (Eq. (10))**.**

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]}\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} + \sum_{t'=t+1}^{T}\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t']}\frac{\partial \mathbf{u}^{l+1}[t']}{\partial \mathbf{s}^l[t']}\frac{\partial \mathbf{s}^l[t']}{\partial \mathbf{u}^l[t']}\prod_{t''=1}^{t'-t}\epsilon^l[t'-t'']. \tag{S3}$$

Proof. According to Fig. 2 and the chain rule, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[T]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[T]}\frac{\partial \mathbf{u}^{l+1}[T]}{\partial \mathbf{s}^l[T]}\frac{\partial \mathbf{s}^l[T]}{\partial \mathbf{u}^l[T]}, \tag{S4}$$

and

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]}\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t+1]}\epsilon^l[t], \tag{S5}$$

when $t = T-1, \cdots, 1$. Eqs. (S4) and (S5) are standard steps in the Backpropagation through time (BPTT) algorithm.

Then we drive Eq. (S3) from Eq. (S5) by induction w.r.t. t. When $t = T-1$,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[T-1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[T-1]}\frac{\partial \mathbf{u}^{l+1}[T-1]}{\partial \mathbf{s}^l[T-1]}\frac{\partial \mathbf{s}^l[T-1]}{\partial \mathbf{u}^l[T-1]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[T]}\epsilon^l[T-1], \tag{S6}$$

*Corresponding author.

which satisfies Eq. (S3). When $t < T-1$, we assume Eq. (S3) is satisfied for $t+1$, then show that Eq. (S3) is satisfied for t:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]} &= \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]} \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t+1]} \epsilon^l[t] \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]} \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \\
&\quad + \left(\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t+1]} \frac{\partial \mathbf{u}^{l+1}[t+1]}{\partial \mathbf{s}^l[t+1]} \frac{\partial \mathbf{s}^l[t+1]}{\partial \mathbf{u}^l[t+1]} \right. \\
&\qquad \left. + \sum_{t'=t+2}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t']} \frac{\partial \mathbf{u}^{l+1}[t']}{\partial \mathbf{s}^l[t']} \frac{\partial \mathbf{s}^l[t']}{\partial \mathbf{u}^l[t']} \prod_{t''=1}^{t'-t-1} \epsilon^l[t'-t''] \right) \epsilon^l[t] \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]} \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t+1]} \frac{\partial \mathbf{u}^{l+1}[t+1]}{\partial \mathbf{s}^l[t+1]} \frac{\partial \mathbf{s}^l[t+1]}{\partial \mathbf{u}^l[t+1]} \epsilon^l[t] \\
&\quad + \sum_{t'=t+2}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t']} \frac{\partial \mathbf{u}^{l+1}[t']}{\partial \mathbf{s}^l[t']} \frac{\partial \mathbf{s}^l[t']}{\partial \mathbf{u}^l[t']} \left(\prod_{t''=1}^{t'-t-1} \epsilon^l[t'-t''] \right) \epsilon^l[t] \\
&= \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t]} \frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} \frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} \\
&\quad + \sum_{t'=t+1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{l+1}[t']} \frac{\partial \mathbf{u}^{l+1}[t']}{\partial \mathbf{s}^l[t']} \frac{\partial \mathbf{s}^l[t']}{\partial \mathbf{u}^l[t']} \prod_{t''=1}^{t'-t} \epsilon^l[t'-t''],
\end{aligned} \tag{S7}$$

where the first equation is due to Eq. (S5), and the second equation is due to the assumption that Eq. (S3) is satisfied for $t+1$. □

C. Time Complexity Analysis of SLTT and BPTT

C.1. Pseudocode of the Bachpropagation Though Time with Surrogate Gradinet Method

We present the pseudocode of one iteration of SNN training with the bachpropagation though time (BPTT) with surrogate gradinet (SG) method in Algorithm S1. Note that the forward pass is defined by

$$\mathbf{u}^l[t] = (1-\frac{1}{\tau})(\mathbf{u}^l[t-1]-V_{th}\mathbf{s}^l[t-1])+\mathbf{W}^l\mathbf{s}^{l-1}[t], \tag{S8}$$

where $\mathbf{s}^l$ are the output spike trains of the l^{th} layer, which are calculated by:

$$\mathbf{s}^l[t] = H(\mathbf{u}^l[t] - V_{th}). \tag{S9}$$

C.2. Time Complexity Analysis

The time complexity of each time step is dominated by the number of scalar multiplication operations. In this subsection, we analyze the required scalar multiplications of the Spatial Learning Through Time (SLTT) and BPTT with SG methods. We show the pseudocode of SLTT in Algorithm S2 again for better presentation.

Algorithm S1 One iteration of SNN training with the BPTT with SG method.

```
Input: Time steps $T$; Network depth $L$; Network parameters $\{\mathbf{W}^l\}_{l=1}^L$; Training data $(\mathbf{s}^0, \mathbf{y})$; Learning rate $\eta$.
 1: //Forward:
 2: for $t = 1, 2, \cdots, T$ do
 3:     for $l = 1, 2, \cdots, L$ do
 4:         Calculate $\mathbf{u}^l[t]$ and $\mathbf{s}^l[t]$ by Eqs. (S8) and (S9);
 5:     end for
 6: end for
 7: Calculate the loss $\mathcal{L}$ based on $\mathbf{s}^L$ and $\mathbf{y}$.
 8: //Backward:
 9: $\mathbf{e}_{\mathbf{s}}^L[T] = \frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[T]}$;
10: for $l = L-1, \cdots, 1$ do
11:     $\mathbf{e}_{\mathbf{u}}^l[T] = \mathbf{e}_{\mathbf{s}}^{l+1}[T]\frac{\partial \mathbf{s}^{l+1}[T]}{\partial \mathbf{u}^l[T]}$, $\mathbf{e}_{\mathbf{s}}^l[T] = \mathbf{e}_{\mathbf{u}}^l[T]\frac{\partial \mathbf{u}^l[T]}{\partial \mathbf{s}^l[T]}$;
12: end for
13: for $t = T-1, T-2, \cdots, 1$ do
14:     for $l = L, L-1, \cdots, 1$ do
15:         if $l = L$ then
16:             $\mathbf{e}_{\mathbf{s}}^L[t] = \frac{\partial \mathcal{L}}{\partial \mathbf{s}^L[t]} + \mathbf{e}_{\mathbf{u}}^L[t+1]\frac{\partial \mathbf{u}^L[t+1]}{\partial \mathbf{s}^L[t]}$;
17:         else
18:             $\mathbf{e}_{\mathbf{s}}^l[t] = \mathbf{e}_{\mathbf{u}}^{l+1}[t]\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]} + \mathbf{e}_{\mathbf{u}}^l[t+1]\frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]}$;
19:         end if
20:         $\mathbf{e}_{\mathbf{u}}^l[t] = \mathbf{e}_{\mathbf{s}}^l[t]\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]} + \mathbf{e}_{\mathbf{u}}^l[t+1]\frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]}$;
21:         $\Delta\mathbf{W}^l$ += $\mathbf{e}_{\mathbf{u}}^l[t]^\top \mathbf{s}^{l-1}[t]^\top$;
22:     end for
23: end for
24: $\mathbf{W}^l = \mathbf{W}^l - \eta\Delta\mathbf{W}^l$, $l = 1, 2, \cdots, L$;
Output: Trained network parameters $\{\mathbf{W}^l\}_{l=1}^L$.
```

Consider that each layer has d neurons. For simplicity, we only consider the scalar multiplications for one intermediate time step ($t < T$) and one intermediate layer ($l < L$). Regarding the BPTT with SG method, it requires scalar multiplications to update $\mathbf{e}_{\mathbf{s}}^l[t]$, $\mathbf{e}_{\mathbf{u}}^l[t]$, and $\Delta\mathbf{W}^l$ at lines 18, 20, and 21 respectively in Algorithm S1. To update $\mathbf{e}_{\mathbf{s}}^l[t]$, two vector-Jacobian products are required. Since $\frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{s}^l[t]}$ is a diagonal matrix, the number of scalar multiplications for updating $\mathbf{e}_{\mathbf{s}}^l[t]$ is d^2+d. To update $\mathbf{e}_{\mathbf{u}}^l[t]$ at line 20, the number is $2d$, since the Jacobians $\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}$ and $\frac{\partial \mathbf{u}^l[t+1]}{\partial \mathbf{u}^l[t]}$ are both diagonal. It requires d^2 scalar multiplications to update $\Delta\mathbf{W}^l$. Regarding the SLTT method, it requires scalar multiplications to update $\mathbf{e}_{\mathbf{u}}^l[t]$ and $\Delta\mathbf{W}^l$ at lines 12 and 13, respectively, in Algorithm S2. It requires d^2+d scalar multiplications to update $\mathbf{e}_{\mathbf{u}}^l[t]$, and requires d^2 to update $\Delta\mathbf{W}^l$. Then compared with the BPTT with SG method, SLTT reduces the number of scalar multiplications by $2d$ for one intermediate time step and one intermediate layer. For SLTT-K, it requires updating $\mathbf{e}_{\mathbf{u}}^l[t]$ and $\Delta\mathbf{W}^l$ only at K randomly chosen time steps. Then the required number of

Algorithm S2 One iteration of SNN training with the SLTT or SLTT-K methods.

```
Input: Time steps $T$; Network depth $L$; Network parameters $\{\mathbf{W}^l\}_{l=1}^L$; Training data $(\mathbf{s}^0, \mathbf{y})$; Learning rate $\eta$; Required backpropagation times $K$ (for SLTT-K).
Initialize: $\Delta\mathbf{W}^l = 0,\ l = 1, 2, \cdots, L$.
1:  if using SLTT-K then
2:      Sample $K$ numbers in $[1, 2, \cdots, T]$ w/o replacement to form $required\_bp\_steps$;
3:  else
4:      $required\_bp\_steps = [1, 2, \cdots, T]$;
5:  end if
6:  for $t = 1, 2, \cdots, T$ do
7:      Calculate $\mathbf{s}^L[t]$ by Eqs. (S8) and (S9);    //Forward
8:      Calculate the instantaneous loss $\ell$;
9:      if $t$ in $required\_bp\_steps$ then    //Backward
10:         $\mathbf{e}_{\mathbf{u}}^L[t] = \frac{1}{T}\frac{\partial \ell}{\partial \mathbf{s}^L[t]}\frac{\partial \mathbf{s}^L[t]}{\partial \mathbf{u}^L[t]}$;
11:         for $l = L-1, \cdots, 1$ do
12:             $\mathbf{e}_{\mathbf{u}}^l[t] = \mathbf{e}_{\mathbf{u}}^{l+1}[t]\frac{\partial \mathbf{u}^{l+1}[t]}{\partial \mathbf{s}^l[t]}\frac{\partial \mathbf{s}^l[t]}{\partial \mathbf{u}^l[t]}$;
13:             $\Delta\mathbf{W}^l \mathrel{+}= \mathbf{e}_{\mathbf{u}}^l[t]^\top \mathbf{s}^{l-1}[t]^\top$;
14:         end for
15:     end if
16: end for
17: $\mathbf{W}^l = \mathbf{W}^l - \eta\Delta\mathbf{W}^l,\ l = 1, 2, \cdots, L$;
Output: Trained network parameters $\{\mathbf{W}^l\}_{l=1}^L$.
```

scalar multiplication operations is proportional to K, which is proportional to T for SLTT and BPTT.

D. Hard Reset Mechanism

In this work, we adopt the LIF model with soft reset as the neuron model, as shown in Eq. (2). Besides the soft reset mechanism, hard reset is also applicable for our method. Specifically, for the LIF model with hard reset

$$\begin{cases} u[t] = (1 - \frac{1}{\tau})v[t-1] + \sum_i w_i s_i[t] + b, \\ s_{out}[t] = H(u[t] - V_{th}), \\ v[t] = u[t] \cdot (1 - s_{out}[t]), \end{cases} \tag{S10}$$

we can also observe that the temporal components contribute a little to $\frac{\partial \mathcal{L}}{\partial \mathbf{u}^l[t]}$ and the observation in Sec. 4.1 still holds. Consider the rectangle surrogate (Eq. (5)) with $\gamma = V_{th}$, the diagonal elements of $\epsilon^l[t]$, which is the dependency between $\mathbf{u}^l[t+1]$ and $\mathbf{u}^l[t]$, become

$$\left(\epsilon^l[t]\right)_{jj} = \begin{cases} \lambda\left(1 - (\mathbf{s}^l[t])_j - \frac{(\mathbf{u}^l[t])_j}{V_{th}}\right), & \frac{1}{2}V_{th} < \left(\mathbf{u}^l[t]\right)_j < \frac{3}{2}V_{th}, \\ \lambda(1 - (\mathbf{s}^l[t])_j), & \text{otherwise.} \end{cases} \tag{S11}$$

Since $(\mathbf{s}^l[t])_j \in \{0, 1\}$, we have $\left(\epsilon^l[t]\right)_{jj} \in (-\frac{3}{2}\lambda, \frac{1}{2}\lambda) \cup \{\lambda\}$, which is at least not large for commonly used small λ. As a result, the spatial components in Eqs. (9) and (10) dominate the gradients and then we can ignore the temporal components without much performance drop.

We conduct experiments with the hard reset mechanism on CIFAR-10 and DVS-CIFAR10, using the same training settings as for soft reset. And the comparison between hard reset and soft reset is shown in Tab. S1. We can see that our method can also achieve competitive results with hard reset.

Table S1: Comparison of accuracy between soft reset and hard reset on CIFAR-10 and DVS-CIFAR10.

Dataset	Reset Mechanism	Acc
CIFAR-10	Soft	$94.44\% \pm 0.21\%$
	Hard	94.34%
DVS-CIFAR10	Soft	$82.20 \pm 0.95\%$
	Hard	81.40%

Table S2: Comparison between SLTT and BPTT on CIFAR-10.

Time Steps	Method	Memory	Time	Acc
$T = 2$	BPTT	1.47G	1.81h	93.90%
	SLTT	**1.10G**	**1.80h**	**93.96%**
$T = 4$	BPTT	2.19G	3.41s	**94.29%**
	SLTT	**1.09G**	**3.29h**	94.17%
$T = 6$	BPTT	3.00G	6.35h	**94.60%**
	SLTT	**1.09G**	**4.58h**	94.59%

E. Performance Under the Smaller T Setting

We compare SLTT and BPTT with SG for smaller T on CIFAR-10. The results are shown in Tab. S2. As T decreases, the behaviors of both methods tend to converge, with SLTT consistently demonstrates better time and memory efficiency than BPTT with SG.

F. Comparison with the SOTA Using the Same Network Architectures

We compare our method with other SOTA using the same network architectures, and the results are shown in Tab. S3. Our method achieves competitive results compared with the SOTA methods, while enabling efficient training at the same time. Furthermore, our method works on different network backbones, showing its effectiveness and applicability.

G. Dataset Description and Preprocessing

CIFAR-10 The CIFAR-10 dataset [16] contains 60,000 32×32 color images in 10 different classes, with 50,000 training samples and 10,000 testing samples. We normal-

Table S3: Comparisons with other SNN training methods using the same network architectures on CIFAR-10, CIFAR-100, ImageNet, DVS-Gesture, and DVS-CIFAR10.

	Method	Network	Time Steps	Efficient Training	Mean$\pm$Std (Best)
CIFAR-10	Dspike[19]	ResNet-18	6	✗	$94.25 \pm 0.07\%$
	TET[7]		6	✗	$94.50 \pm 0.07\%$
	DSR [21]		20	✓	$95.40 \pm 0.15\%$
	RecDis[13]		6	✗	$\mathbf{95.55 \pm 0.05}\%$
	SLTT (ours)		6	✓	$94.44\% \pm 0.21\%$ (94.59%)
	LTL-Online [32] [1]	VGG-16	16	✓	92.85%
	DIET-SNN [25] [1]		10	✗	93.44%
	Temporal Pruning [5]		5	✗	$\mathbf{93.90}\%$
	SLTT (ours)		6	✓	$93.28\% \pm 0.02\%$ (93.29%)
CIFAR-100	RecDis[13]	ResNet-18	4	✗	$74.10 \pm 0.13\%$
	TET[7]		6	✗	$74.72 \pm 0.28\%$
	DSR [21]		20	✓	$\mathbf{78.50 \pm 0.12}\%$
	Dspike[19]		6	✗	$74.24 \pm 0.10\%$
	SLTT (ours)		6	✓	$74.38\% \pm 0.30\%$ (74.67%)
	ANN-to-SNN[4] [1]	VGG-16	8	✓	$\mathbf{73.96}\%$
	DIET-SNN [25] [1]		10	✗	69.67%
	Temporal Pruning [5]		5	✗	71.58%
	SLTT (ours)		6	✓	$\underline{72.55\% \pm 0.24\% \; (72.83\%)}$
ImageNet	ANN-to-SNN [22] [1]	ResNet-34	256, 512	✓	$73.16\%, 74.18\%$
	ANN-to-SNN[18] [1]		32, 256	✓	$64.54\%, \mathbf{74.61\%}$
	TET[7]		6	✗	64.79%
	OTTT[30]		6	✓	65.15%
	SEW [10]		4	✗	67.04%
	SLTT (ours)		6	✓	66.19%
	STBP-tdBN [33]	ResNet-50	6	✗	64.88%
	SEW [10]		4	✗	67.78%
	ANN-to-SNN [22] [1]		256, 512	✓	$73.56\%, \mathbf{75.04}\%$
	SLTT (ours)		6	✓	67.02%
	ANN-to-SNN [22] [1]	ResNet-101	256, 512	✓	$\mathbf{73.50}\%, \mathbf{75.72}\%$
	SEW [10]		4	✗	68.76%
	SLTT-2 (ours)		6	✓	69.26%
DVS-Gesture	STBP-tdBN [33]	ResNet-18	40	✗	96.87%
	SLTT (ours)		20	✓	$\mathbf{97.68 \pm 0.53}\%$ $(\mathbf{98.26}\%)$
	OTTT [30]	VGG-11	20	✓	96.88%
	SLTT (ours)		20	✓	$\mathbf{98.50 \pm 0.21}\%$ $(\mathbf{98.62}\%)$
DVS-CIFAR10	STBP-tdBN[33]	ResNet-18	10	✗	67.80%
	Dspike[19]		10	✗	$75.40 \pm 0.05\%$
	InfLoR[12]		10	✗	$75.50 \pm 0.12\%$
	SLTT (ours)		10	✓	$\mathbf{81.87 \pm 0.49}\%$ $(\mathbf{82.20}\%)$
	DSR [21]	VGG-11	20	✓	$77.27 \pm 0.24\%$
	OTTT [30]		10	✓	$76.27 \pm 0.05\%(76.30\%)$
	TET[7]		10	✗	$\mathbf{83.17 \pm 0.15}\%$
	SLTT (ours)		10	✓	$82.20 \pm 0.95\%$ (83.10%)

[1] Pre-trained ANN models are required.

ize the image data to ensure that input images have zero mean and unit variance. We apply random cropping with 4 padding on each border of the image, random horizontal flipping, and cutout [8] for data augmentation. We apply direct encoding [25] to encode the image pixels into time series. Specifically, the pixel values are repeatedly applied to the input layer at each time step. CIFAR-10 is licensed under MIT.

CIFAR-100 The CIFAR-100 dataset [16] is similar to CIFAR-10 except that it contains 100 classes of objects.

There are 50,000 training samples and 10,000 testing samples, each of which is a 32×32 color image. CIFAR-100 is licensed under MIT. We adopt the same data preprocessing and input encoding as CIFAR-10.

ImageNet The ImageNet-1K dataset [6] contains color images in 1000 classes of objects, with 1,281,167 training images and 50,000 validation images. This dataset is licensed under Custom (non-commercial). We normalize the image data to ensure that input images have zero mean and unit variance. For training samples, we apply random resized cropping to get images with size 224×224, and then apply horizontal flipping. For validation samples, we resize the images to 256×256 and then center-cropped them to 224×224. We transform the pixels into time sequences by direct encoding [25], as done for CIFAR-10 and CIFAR-100.

DVS-Gesture The DVS-Gesture [1] dataset is recorded using a Dynamic Vision Sensor (DVS), consisting of spike trains with two channels corresponding to ON- and OFF-event spikes. The dataset contains 11 hand gestures from 29 subjects under 3 illumination conditions, with 1176 training samples and 288 testing samples. The license of DVS-Gesture is Creative Commons Attribution 4.0. For data preprocessing, we follow [11] to integrate the events into frames. The event-to-frame integration is handled with the SpikingJelly [9] framework.

DVS-CIFAR10 The DVS-CIFAR10 dataset [17] is a neuromorphic dataset converted from CIFAR-10 using a DVS camera. It contains 10,000 event-based images with pixel dimensions expanded to 128×128. The dataset is licensed under CC BY 4.0. We split the whole dataset into 9000 training images and 1000 testing images. Regarding data preprocessing, we integrate the events into frames [11], and reduce the spatial resolution into 48×48 by interpolation. For some experiments, we take random horizontal flip and random roll within 5 pixels as data augmentation, the same as [7].

H. Network Architectures

H.1. Scaled Weight Standardization

An important characteristic of the proposed SLTT method is the instantaneous gradient calculation at each time step, which enables time-steps-independent memory costs. Under our instantaneous update framework, an effective technique, batch normalization (BN) along the temporal dimension [33, 19, 7, 21], cannot be adopted to our method, since this technique requires gathering data from all time steps to calculate the mean and variance statistics. For some tasks, we still use BN components, but their calculated statistics are based on data from each time step. For other tasks, we replace BN with scaled weight standardization (sWS) [24, 2, 3], as introduced below.

The sWS component [2], which is modified from the original weight standardization [24], normalizes the weights according to:

$$\hat{\mathbf{W}}_{i,j} = \gamma \frac{\mathbf{W}_{i,j} - \mu_{\mathbf{W}_{i,\cdot}}}{\sigma_{\mathbf{W}_{i,\cdot}}}, \tag{S12}$$

where the mean $\mu_{\mathbf{W}_{i,\cdot}}$ and standard deviation $\sigma_{\mathbf{W}_{i,\cdot}}$ are calculated across the fan-in extent indexed by i, N is the dimension of the fan-in extent, and γ is a fixed hyperparameter. The hyperparameter γ is set to stabilize the signal propagation in the forward pass. Specifically, for one network layer $\mathbf{z} = \hat{\mathbf{W}} g(\mathbf{x})$, where $g(\cdot)$ is the activation and the elements of $\mathbf{x}$ are i.i.d. from $\mathcal{N}(0,1)$, we determine γ to make $\mathbb{E}(\mathbf{z}) = 0$, and $\mathrm{Cov}(\mathbf{z}) = \mathbf{I}$. For SNNs, the activation $g(\cdot)$ at each time step can be treated as the Heaviside step function. Then we take $\gamma \approx 2.74$ to stable the forward propagation, as calculated by [30]. Furthermore, sWS incorporate another learnable scaling factor for the weights [2, 30] to mimic the scaling factor of BN. sWS shares similar effects with BN, but introduces no dependence of data from different batches and time steps. Therefore, sWS is a convincing alternative for replacing BN in our framework.

H.2. Normalization-Free ResNets

For deep ResNets [14], sWS cannot enjoy the similar signal-preserving property as BN very well due to the skip connection. Then in some experiments, we consider the normalization-free ResNets (NF-ResNets) [2, 3] that not only replace the BN components by sWS but also introduce other techniques to preserve the signal in the forward pass.

The NF-ResNets use the residual blocks of the form $x_{l+1} = x_l + \alpha f_l\left(x_l/\beta_l\right)$, where α and β_l are hyperparameters used to stabilize signals, and the weights in $f_l(\cdot)$ are imposed with sWS. The carefully determined β_l and the sWS component together ensure that $f_l(x_l/\beta_l)$ has unit variance. α controls the rate of variance growth between blocks, and is set to be 0.2 in our experiments. Please refer to [2] for more details on the network design.

H.3. Description of Adopted Network Architectures

We adopt ResNet-18 [14] with the pre-activation residual blocks [15] to conduct experiments on CIFAR-10 and CIFAR-100. The channel sizes for the four residual blocks are 64, 128, 256, and 512, respectively. All the ReLU activations are substituted by the leaky integrate and fire (LIF) neurons. To make the network implementable for neuromorphic computing, we replace all the max pooling operations with average pooling. To enable instantaneous gradient calculation, we adopt the BN components that calculate the mean and variance statistics for each time step, not the total time horizon. Then for each iteration, a BN component is implemented for T times, where T is the number of total time steps.

We adopt VGG-11 [28] to conduct experiments on DVS-Gesture and DVS-CIFAR10. As done for Resnet-18, we substitute all the max poolings with average poolings and use the time-step-wise BN. We remove two fully connected layers [21, 7, 30] to reduce the computation. A dropout layer [29] is added behind each LIF neuron layer to bring better generalization. The dropout rates for DVS-Gesture and DVS-CIFAR10 are set to be 0.4 and 0.3, respectively.

We also adopt VGG-11 (WS) to conduct experiments on DVS-Gesture and achieve state-of-the-art performance. VGG-11 (WS) is a BN-free network that shares a similar architecture with VGG-11 introduced above. The difference between the two networks is that VGG-11 consists of the convolution-BN-LIF blocks while VGG-11 (WS) consists of the convolution-sWS-LIF blocks.

We adopt NF-ResNet-34, NF-ResNet-50, and NF-ResNet-101 to conduct experiments on ImageNet. Those networks are normalization-free ResNets introduced in Appendix H.2. In Figs. 1 and 3 of the main content, the used network for ImageNet is NF-ResNet-34.

I. Training Settings

All the implementation is based on the PyTorch [23] and SpikingJelly [9] frameworks, and the experiments are carried out on one Tesla-V100 GPU or one Tesla-A100 GPU.

For CIFAR-10, CIFAR-100, and DVS-Gesture, we adopt the loss function proposed in [7]:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}(1-\lambda)\ell_1(\mathbf{o}[t], y) + \lambda\ell_2(\mathbf{o}[t], V_{th}), \quad \text{(S13)}$$

where T is the number of total time steps, ℓ_1 is the cross entropy function, ℓ_2 is the mean squared error (MSE) function, $\mathbf{o}[t]$ is the network output at the t-th time step, y is the label, λ is a hyperparameter taken as 0.05, and V_{th} is the spike threshold which is set to be 1 in this work. For ImageNet, we use the same loss but simply set $\lambda = 0$. For DVS-CIFAR10, we also combine the cross entropy loss and the MSE loss, but the MSE loss does not act as a regularization term as in [7]:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}(1-\lambda)\ell_1(\mathbf{o}[t], y) + \lambda\ell_2(\mathbf{o}[t], y), \quad \text{(S14)}$$

where α is also taken as 0.05. Our experiments show that such loss performs better than that defined in Eq. (S13) for DVS-CIFAR10.

Table S4: Training hyperparameters about optimization. "WD" means weight decay, "LR" means initial learning rate, and "BS" means batch size.

Dataset	Epoch	LR	BS	WD
CIFAR-10	200	0.1	128	5×10^{-5}
CIFAR-100	200	0.1	128	5×10^{-4}
ImageNet (pre-train)	100	0.1	256	1×10^{-5}
ImageNet (fine-tune)	30	0.001	256	0
DVS-Gesture	300	0.1	16	5×10^{-4}
DVS-CIFAR10	300	0.05	128	5×10^{-4}

For all the tasks, we use SGD [27] with momentum 0.9 to train the networks, and use cosine annealing [20] as the learning rate schedule. Other hyperparameters about optimization are listed in Tab. S4. For ImageNet, we first train the SNN with only 1 time step to get a pre-trained model, and then fine-tune the model for multiple time steps.

In Section. 5.3 of the main content, we compare the proposed SLTT method and OTTT [30] following the same experimental settings as introduced in [30]. For both methods, we adopt the NF-ResNet-34 architecture for ImageNet and VGG-11 (WS) for other datasets. And the total number of time steps for CIFAR-10, CIFAR-100, ImageNet, DVS-Gesture, and DVS-CIFAR10 are 6, 6, 6, 20, and 10, respectively.

J. Societal impact and limitations

There is no direct negative societal impact since this work focuses on efficient training methods for SNNs. Compared with ANNs, the inference of SNNs requires less energy consumption and produces less carbon dioxide emissions. Our proposed method can further reduce energy consumption in the SNN training process. As for the limitation, the SLTT method cannot be equipped with some network techniques, such as batch normalization along the temporal dimension, since the proposed method is conducted in an online manner, which is biologically more plausible and more friendly for on-chip training. It may require the exploration of more techniques that are compatible with online learning of SNNs.

References

[1] 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 *CVPR*, 2017. 5

[2] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. In *ICLR*, 2021. 5

[3] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition

without normalization. In *International Conference on Machine Learning*, pages 1059–1071. PMLR, 2021. 5

[4] Tong Bu, Wei Fang, Jianhao Ding, Penglin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *ICLR*, 2022. 4

[5] Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. Towards ultra low latency spiking neural networks for vision and sequential tasks using temporal pruning. In *ECCV*, 2022. 4

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

[7] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *ICLR*, 2022. 4, 5, 6

[8] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 4

[9] Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. Spikingjelly. `https://github.com/fangwei123456/spikingjelly`, 2020. 5, 6

[10] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *NeurIPS*, 2021. 4

[11] 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 *ICCV*, 2021. 5

[12] Yufei Guo, Yuanpei Chen, Liwen Zhang, YingLei Wang, Xiaode Liu, Xinyi Tong, Yuanyuan Ou, Xuhui Huang, and Zhe Ma. Reducing information loss for spiking neural networks. In *ECCV*, 2022. 4

[13] Yufei Guo, Xinyi Tong, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Zhe Ma, and Xuhui Huang. Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In *CVPR*, 2022. 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 5

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3, 4

[17] 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. 5

[18] Yuhang Li, Shikuang Deng, Xin Dong, Ruihao Gong, and Shi Gu. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *ICML*, 2021. 4

[19] Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. In *NeurIPS*, 2021. 4, 5

[20] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 6

[21] 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 *CVPR*, 2022. 1, 4, 5, 6

[22] 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. 4

[23] 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 *NeurIPS*, 2019. 6

[24] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Micro-batch training with batch-channel normalization and weight standardization. *arXiv preprint arXiv:1903.10520*, 2019. 5

[25] Nitin Rathi and Kaushik Roy. Diet-snn: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *TNNLS*, 2021. 4, 5

[26] 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. 1

[27] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 6

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014. 6

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 6

[30] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. In *NeurIPS*, 2022. 4, 5, 6

[31] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. In *NeurIPS*, 2021. 1

[32] Qu Yang, Jibin Wu, Malu Zhang, Yansong Chua, Xinchao Wang, and Haizhou Li. Training spiking neural networks with local tandem learning. *arXiv preprint arXiv:2210.04532*, 2022. 4

[33] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *AAAI*, 2021. 1, 4, 5