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

Qingyan Meng\textsuperscript{1,2}, Mingqing Xiao\textsuperscript{3}, Shen Yan\textsuperscript{4}, Yisen Wang\textsuperscript{3,5}, Zhouchen Lin\textsuperscript{3,5,6,*}, Zhi-Quan Luo\textsuperscript{1,2}
\textsuperscript{1}The Chinese University of Hong Kong, Shenzhen \textsuperscript{2}Shenzhen Research Institute of Big Data \textsuperscript{3}Key Lab. of Machine Perception (MoE), School of Artificial Intelligence, Peking University \textsuperscript{4}Center for Data Science, Peking University \textsuperscript{5}Institute for Artificial Intelligence, Peking University \textsuperscript{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:

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

where \( l = 1, \ldots, L \) is the layer index, \( t = 1, \ldots, T \) is the time step index, \( \{\mathbf{s}^l[t]\}^{T}_{t=1} \) are the input to the network, and \( \{\mathbf{s}^l[t]\}^{T}_{t=1} \) are the binary output sequence of the \( l \)-th layer. \( \tau \) is a pre-defined time constant, \( H(\cdot) \) is 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 matrices to be trained. \( \mathbf{v}^l[0] \) is set to be 0 for \( l = 1, \ldots, L \).

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 \( \{\mathbf{s}^L[t]\}^{T}_{t=1} \) 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 \( \{\mathbf{s}^0[t]\}^{T}_{t=1} \) is classified as belonging to the \( i \)-th class if \( \mathbf{o}_i \) is the largest value among \( \{\mathbf{o}_1, \ldots, \mathbf{o}_c\} \).

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

\[
\epsilon'[t] = \frac{\partial \mathbf{u}^t[t + 1]}{\partial \mathbf{u}^t[t]} + \frac{\partial \mathbf{u}^t[t + 1]}{\partial \mathbf{s}^t[t]} \frac{\partial \mathbf{s}^t[t]}{\partial \mathbf{u}^t[t]},
\] (S2)

which is the dependency between \( \mathbf{u}^t[t + 1] \) and \( \mathbf{u}^t[t] \). 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)).

\[
\frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[t]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{t+1}[T]} \frac{\partial \mathbf{u}^{t+1}[T]}{\partial \mathbf{s}^t[t]} \frac{\partial \mathbf{s}^t[t]}{\partial \mathbf{u}^t[t]} + \sum_{t' = t+1}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{t+1}[t']} \frac{\partial \mathbf{u}^{t+1}[t']}{\partial \mathbf{s}^t[t]} \frac{\partial \mathbf{s}^t[t]}{\partial \mathbf{u}^t[t]} \prod_{t'' = t'}^{t'-1} \epsilon'[t'' - t'],
\] (S3)

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

\[
\frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[T]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{t+1}[T]} \frac{\partial \mathbf{u}^{t+1}[T]}{\partial \mathbf{s}^t[T]} \frac{\partial \mathbf{s}^t[T]}{\partial \mathbf{u}^t[T]},
\] (S4)

and

\[
\frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[t]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{t+1}[t]} \frac{\partial \mathbf{u}^{t+1}[t]}{\partial \mathbf{s}^t[t]} \frac{\partial \mathbf{s}^t[t]}{\partial \mathbf{u}^t[t]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[t + 1]} \epsilon'[t],
\] (S5)

when \( t = T - 1, \ldots, 1 \). 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 \( t = T - 1 \),

\[
\frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[T - 1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{t+1}[T - 1]} \frac{\partial \mathbf{u}^{t+1}[T - 1]}{\partial \mathbf{s}^t[T - 1]} \frac{\partial \mathbf{s}^t[T - 1]}{\partial \mathbf{u}^t[T - 1]} + \frac{\partial \mathcal{L}}{\partial \mathbf{u}^t[T]} \epsilon'[T - 1],
\] (S6)
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 \( t \):

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

\[
\frac{\partial L}{\partial u^{l+1}[t]} \frac{\partial u^{l+1}[t]}{\partial s^l[t]} \frac{\partial s^l[t]}{\partial u^l[t]} + \left( \frac{\partial L}{\partial u^{l+1}[t+1]} \frac{\partial u^{l+1}[t+1]}{\partial s^l[t+1]} \frac{\partial s^l[t+1]}{\partial u^l[t+1]} + \sum_{t'=t+2}^{T} \frac{\partial L}{\partial u^{l+1}[t']} \frac{\partial u^{l+1}[t']} {\partial s^l[t']} \frac{\partial s^l[t']} {\partial u^l[t']} \prod_{t''=t+1}^{t'-1} \epsilon^l[t'-t''] \right) \epsilon^l[t] \]

\[
\frac{\partial L}{\partial u^{l+1}[t] \frac{\partial u^{l+1}[t]}{\partial s^l[t]} \frac{\partial s^l[t]}{\partial u^l[t]} + \frac{\partial L}{\partial u^l[t+1]} \frac{\partial u^{l+1}[t+1]}{\partial s^l[t+1]} \frac{\partial s^l[t+1]}{\partial u^l[t+1]} + \sum_{t'=t+1}^{T} \frac{\partial L}{\partial u^{l+1}[t']} \frac{\partial u^{l+1}[t']} {\partial s^l[t']} \frac{\partial s^l[t']} {\partial u^l[t']} \prod_{t''=t+1}^{t'-1} \epsilon^l[t'-t''] \right) \epsilon^l[t] \]

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 backpropagation though time (BPTT) with surrogate gradient (SG) method in Algorithm S1. Note that the forward pass is defined by

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

where \( s^l \) are the output spike trains of the \( l \)th layer, which are calculated by:

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

C.2. Time Complexity Analysis

The time complexity of each time step is dominated by the number of scalar multiplications. 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 \( \{W^l\}_{l=1}^L \); Training data \((s^0, y)\); Learning rate \( \eta \).

1. **//Forward:**
2. for \( t = 1, 2, \cdots, T \) do
3. for \( l = 1, 2, \cdots, L \) do
4. Calculate \( u^l[t] \) and \( s^l[t] \) by Eqs. (S8) and (S9);
5. end for
6. end for
7. Calculate the loss \( L \) based on \( s^L \) and \( y \).
8. **//Backward:**
9. for \( l = L - 1, \cdots, 1 \) do
10. \( e^L_s[T] = \frac{\partial L}{\partial s^L[T]} \);
11. \( e^l_s[T] = e^{l+1}_s[T] \frac{\partial u^{l+1}[t]}{\partial s^l[t]}, e^l_s[T] = e^l_s[T] \frac{\partial u^l[t]}{\partial s^l[t]} \);
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. \( e^L_s[t] = \frac{\partial L}{\partial s^L[t]} + e^L_u[t+1] \frac{\partial u^l[t]}{\partial s^l[t]} \);
17. else
18. \( e^l_s[t] = e^{l+1}_u[t] \frac{\partial u^{l+1}[t]}{\partial s^l[t]} + e^l_u[t+1] \frac{\partial u^l[t]}{\partial s^l[t]} \);
19. end if
20. \( \Delta W^l = e^l_s[t] \ s^{l-1}[t] \);
21. end for
22. end for
23. end for
24. \( W^l = W^l - \eta \Delta W^l, \ l = 1, 2, \cdots, L; \)

**Output:** Trained network parameters \( \{W^l\}_{l=1}^L \).
Algorithm S2 One iteration of SNN training with the SLTT or SLTT-K methods.

**Input:** Time steps \(T\); Network depth \(L\); Network parameters \(\{W^l\}_{l=1}^L\); Training data \((s^0, y)\); Learning rate \(\eta\); Required backpropagation times \(K\) (for SLTT-K).

**Initialize:** \(\Delta W^l_l = 0, l = 1, 2, \cdots, L\).

1: if using SLTT-K then
2: Sample \(K\) numbers in \([1, 2, \cdots, T]\) w/o replacement to form \(\text{required\_bp\_steps}\);
3: else
4: \(\text{required\_bp\_steps} = [1, 2, \cdots, T]\);
5: end if
6: for \(t = 1, 2, \cdots, T\) do
7: Calculate \(s^l[t]\) by Eqs. (S8) and (S9); //Forward
8: Calculate the instantaneous loss \(\ell\);
9: if in \(\text{required\_bp\_steps}\) then //Backward
10: \(e^l_{u}[t] = \frac{\partial \ell}{\partial u^l[t]} \partial u^l[t];
11: for \(l = L - 1, \cdots, 1\) do
12: \(e^l_{u}[t] = e^l_{u}[t] + \frac{\partial \ell}{\partial u^l[t]} \partial u^l[t];
13: \(\Delta W^l_l += e^l_{u}[t] s^{l-1}[t]^t;\)
14: end for
15: end if
16: end for
17: \(W^l_l = W^l_l - \eta \Delta W^l_l, l = 1, 2, \cdots, L;\)

**Output:** Trained network parameters \(\{W^l\}_{l=1}^L\).

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

\[
\begin{align*}
    u[t] &= (1 - \frac{1}{\tau_v}) u[t - 1] + \sum_i w_i s_i[t] + b, \\
    s_{out}[t] &= H(u[t] - V_h), \\
    v[t] &= u[t] \cdot (1 - s_{out}[t]),
\end{align*}
\]  

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

\[
\left(e^l_{u}[t]\right)_{jj} = \begin{cases} 
\lambda \left(1 - (s^l[t])_j - \frac{(u^l[t])[j]}{V_h}\right), & \frac{1}{2} V_h < (u^l[t])[j] < \frac{3}{2} V_h, \\
\frac{3}{2} V_h, & \text{otherwise}.
\end{cases}
\]

Since \((s^l[t])[j] \in \{0, 1\}\), we have \(\left(e^l_{u}[t]\right)_{jj} \in \left(-\frac{3}{2} \lambda, \frac{3}{2} \lambda\right) \cup \{\lambda\}\), which is at least not large for commonly used small \(\lambda\). 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.

### 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 demonstrating 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 in 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Network</th>
<th>Time Steps</th>
<th>Efficient Training</th>
<th>Mean±Std (Best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dspike [19]</td>
<td>ResNet-18</td>
<td>6</td>
<td>X</td>
<td>94.25±0.07%</td>
</tr>
<tr>
<td>TET [7]</td>
<td>ResNet-18</td>
<td>6</td>
<td>X</td>
<td>94.50±0.07%</td>
</tr>
<tr>
<td>DSR [21]</td>
<td>ResNet-18</td>
<td>20</td>
<td>✓</td>
<td>95.40±0.15%</td>
</tr>
<tr>
<td>RecDis [13]</td>
<td>ResNet-18</td>
<td>6</td>
<td>X</td>
<td>95.55±0.05%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-18</td>
<td>6</td>
<td>✓</td>
<td>94.44±0.21% (94.59%)</td>
</tr>
<tr>
<td>LTL-Online [32]</td>
<td>16</td>
<td>✓</td>
<td></td>
<td>92.85%</td>
</tr>
<tr>
<td>DIET-SNN [25]</td>
<td>VGG-16</td>
<td>10</td>
<td>X</td>
<td>93.44%</td>
</tr>
<tr>
<td>Temporal Pruning [5]</td>
<td>5</td>
<td>✓</td>
<td></td>
<td>93.90%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-18</td>
<td>6</td>
<td>✓</td>
<td>93.28% ± 0.02% (93.29%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Network</th>
<th>Time Steps</th>
<th>Efficient Training</th>
<th>Mean±Std (Best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecDis [13]</td>
<td>ResNet-18</td>
<td>4</td>
<td>X</td>
<td>74.10±0.13%</td>
</tr>
<tr>
<td>TET [7]</td>
<td>ResNet-18</td>
<td>6</td>
<td></td>
<td>74.72±0.28%</td>
</tr>
<tr>
<td>DSR [21]</td>
<td>ResNet-18</td>
<td>20</td>
<td>✓</td>
<td>78.50±0.12%</td>
</tr>
<tr>
<td>Dspike [19]</td>
<td>ResNet-18</td>
<td>6</td>
<td>X</td>
<td>74.24±0.10%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-18</td>
<td>6</td>
<td>✓</td>
<td>74.38% ± 0.39% (74.67%)</td>
</tr>
<tr>
<td>ANN-to-SNN [6]</td>
<td>8</td>
<td>✓</td>
<td></td>
<td>73.96%</td>
</tr>
<tr>
<td>DIET-SNN [25]</td>
<td>VGG-16</td>
<td>10</td>
<td>X</td>
<td>69.67%</td>
</tr>
<tr>
<td>Temporal Pruning [5]</td>
<td>5</td>
<td>✓</td>
<td></td>
<td>71.58%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>VGG-16</td>
<td>6</td>
<td>✓</td>
<td>72.55% ± 0.24% (72.83%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Network</th>
<th>Time Steps</th>
<th>Efficient Training</th>
<th>Mean±Std (Best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN-to-SNN [22]</td>
<td>256,512</td>
<td>✓</td>
<td></td>
<td>73.16%, 74.18%</td>
</tr>
<tr>
<td>ANN-to-SNN [18]</td>
<td>32,256</td>
<td>✓</td>
<td></td>
<td>64.54%, 74.61%</td>
</tr>
<tr>
<td>TET [7]</td>
<td>ResNet-34</td>
<td>6</td>
<td>X</td>
<td>64.79%</td>
</tr>
<tr>
<td>OTTT [30]</td>
<td>ResNet-50</td>
<td>6</td>
<td>✓</td>
<td>65.15%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-101</td>
<td>6</td>
<td>✓</td>
<td>66.19%</td>
</tr>
<tr>
<td>STBP-tdBN [33]</td>
<td>ResNet-18</td>
<td>6</td>
<td>X</td>
<td>64.88%</td>
</tr>
<tr>
<td>SEW [10]</td>
<td>ResNet-50</td>
<td>4</td>
<td>X</td>
<td>67.78%</td>
</tr>
<tr>
<td>ANN-to-SNN [22]</td>
<td>256,512</td>
<td>✓</td>
<td></td>
<td>73.56%, 75.04%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-101</td>
<td>6</td>
<td>✓</td>
<td>67.02%</td>
</tr>
<tr>
<td>ANN-to-SNN [22]</td>
<td>256,512</td>
<td>✓</td>
<td></td>
<td>73.50%, 75.72%</td>
</tr>
<tr>
<td>SEW [10]</td>
<td>ResNet-101</td>
<td>4</td>
<td>X</td>
<td>68.76%</td>
</tr>
<tr>
<td>SLTT-2 (ours)</td>
<td>ResNet-101</td>
<td>6</td>
<td>✓</td>
<td>69.26%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Network</th>
<th>Time Steps</th>
<th>Efficient Training</th>
<th>Mean±Std (Best)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STBP-tdBN [33]</td>
<td>ResNet-18</td>
<td>40</td>
<td>X</td>
<td>96.87%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-18</td>
<td>20</td>
<td>✓</td>
<td>97.68±0.53% (98.26%)</td>
</tr>
<tr>
<td>OTTT [30]</td>
<td>VGG-11</td>
<td>20</td>
<td>✓</td>
<td>96.88%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>VGG-11</td>
<td>20</td>
<td>✓</td>
<td>98.50±0.21% (98.62%)</td>
</tr>
<tr>
<td>STBP-tdBN [33]</td>
<td>ResNet-18</td>
<td>10</td>
<td>X</td>
<td>67.80%</td>
</tr>
<tr>
<td>Dspike [19]</td>
<td>ResNet-18</td>
<td>10</td>
<td>✓</td>
<td>75.40±0.05%</td>
</tr>
<tr>
<td>InfLoR [12]</td>
<td>ResNet-18</td>
<td>10</td>
<td>X</td>
<td>75.50±0.12%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>ResNet-18</td>
<td>10</td>
<td>✓</td>
<td>81.87±0.49% (82.20%)</td>
</tr>
<tr>
<td>DSR [21]</td>
<td>VGG-11</td>
<td>20</td>
<td>✓</td>
<td>77.27±0.24%</td>
</tr>
<tr>
<td>OTTT [30]</td>
<td>VGG-11</td>
<td>10</td>
<td>✓</td>
<td>76.27±0.05% (76.30%)</td>
</tr>
<tr>
<td>TET [7]</td>
<td>VGG-11</td>
<td>10</td>
<td>X</td>
<td>83.17±0.15%</td>
</tr>
<tr>
<td>SLTT (ours)</td>
<td>VGG-11</td>
<td>10</td>
<td>✓</td>
<td>82.20±0.95% (83.10%)</td>
</tr>
</tbody>
</table>

1 Pre-trained ANN models are required.

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 \(32 \times 32\) 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 \(224 \times 224\), and then apply horizontal flipping. For validation samples, we resize the images to \(256 \times 256\) and then center-cropped them to \(224 \times 224\). 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 \(128 \times 128\). 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 \(48 \times 48\) 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:

\[
\hat{W}_{i,j} = \gamma \frac{W_{i,j} - \mu_{W_i}}{\sigma_{W_i}},
\]

where the mean \(\mu_{W_i}\) and standard deviation \(\sigma_{W_i}\) are calculated across the fan-in extent indexed by \(i\), \(N\) is the dimension of the fan-in extent, and \(\gamma\) is a fixed hyperparameter. The hyperparameter \(\gamma\) is set to stabilize the signal propagation in the forward pass. Specifically, for one network layer \(z = Wg(x)\), where \(g(\cdot)\) is the activation and the elements of \(x\) are i.i.d. from \(N(0, 1)\), we determine \(\gamma\) to make \(\mathbb{E}(z) = 0\), and \(\text{Cov}(z) = \mathbb{I}\). 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 \(x_{t+1} = x_t + \alpha f_t(x_t/\beta_t)\), where \(\alpha\) and \(\beta_t\) are hyperparameters used to stabilize signals, and the weights in \(f_t(\cdot)\) are imposed with sWS. The carefully determined \(\beta_t\) and the sWS component together ensure that \(f_t(x_t/\beta_t)\) has unit variance. \(\alpha\) 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 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]:

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

where $T$ is the number of total time steps, $\ell_1$ is the cross entropy function, $\ell_2$ is the mean squared error (MSE) function, $o[t]$ is the network output at the $t$-th time step, $y$ is the label, $\lambda$ 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]:

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

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Epoch</th>
<th>LR</th>
<th>BS</th>
<th>WD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td>200</td>
<td>0.1</td>
<td>128</td>
<td>$5 \times 10^{-5}$</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>200</td>
<td>0.1</td>
<td>128</td>
<td>$5 \times 10^{-4}$</td>
</tr>
<tr>
<td>ImageNet (pre-train)</td>
<td>100</td>
<td>0.1</td>
<td>256</td>
<td>$1 \times 10^{-5}$</td>
</tr>
<tr>
<td>ImageNet (fine-tune)</td>
<td>30</td>
<td>0.001</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>DVS-Gesture</td>
<td>300</td>
<td>0.1</td>
<td>16</td>
<td>$5 \times 10^{-4}$</td>
</tr>
<tr>
<td>DVS-CIFAR10</td>
<td>300</td>
<td>0.05</td>
<td>128</td>
<td>$5 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

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

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
