Supplementary Materials for: Training High-Performance Low-Latency Spiking Neural Networks by Differentiation on Spike Representation

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. Details about Spike Representation

A.1. Derivation for Eq. (14)

In this subsection, we consider the LIF model defined by Eqs. (3a) to (3c) and (5), and derive Eq. (14) from Eq. (13) in the main content under mild assumptions.

From the main content, we have derived that

$$\hat{a}[N] \approx \frac{\hat{I}[N]}{\tau} - \frac{V[N]}{\Delta t \sum_{n=1}^{N} \lambda^{N-n}}, \tag{S1}$$

as shown in Eq. (13). Since the LIF neuron is supposed to fire no or very few spikes when $\frac{\hat{I}[N]}{\tau} < 0$ and fire almost always when $\frac{\hat{I}[N]}{\tau} > \frac{V_{th}}{\Delta t}$, we can separate the accumulated membrane potential $V[N]$ into two parts: one part $V^-[N]$ represents the "exceeded" membrane potential that does not contribute to spike firing, and the other part $V^+[N]$ represents the "remaining" membrane potential. In detail, the "exceeded" membrane potential $V^-[N]$ can be calculated as

$$V^-[N] = \begin{cases} (\Delta t \sum_{n=1}^{N} \lambda^{N-n}) \frac{\hat{I}[N]}{\tau}, & \frac{\hat{I}[N]}{\tau} < 0, \\ (\Delta t \sum_{n=1}^{N} \lambda^{N-n})(\frac{\hat{I}[N]}{\tau} - \frac{V_{th}}{\Delta t}), & \frac{\hat{I}[N]}{\tau} > \frac{V_{th}}{\Delta t}, \\ 0, & \text{otherwise.} \end{cases} \tag{S2}$$

And the "remaining" membrane potential can be calculated as $V^+[N] = V[N] - V^-[N]$. With the decomposition of membrane potential $V[N]$ and the fact that $\Delta t \sum_{n=1}^{N} \lambda^{N-n} = \tau$ when $N \to \infty$ and $\Delta t \to 0$, we can further approximate $\hat{a}[N]$ from Eq. (S1) as

$$\lim_{N\to\infty} \hat{a}[N] \approx \lim_{N\to\infty} \text{clamp}\left(\frac{\hat{I}[N]}{\tau} - \frac{V^+[N]}{\tau}, 0, \frac{V_{th}}{\Delta t}\right), \tag{S3}$$

if the limit of right hand side exists.

*Corresponding author.

Now we want to find the condition to ignore the term $\frac{V^+[N]}{\tau}$ in Eq. (S3). In the case $V^-[N] \neq 0$, the magnitude of membrane potential $|V(n)|$ would gradually increase with time. After introducing V^-, the "remaining" membrane potential $V^+[N]$ typically does not diverge over time. In fact, $V^+[N]$ is typically bounded in $[0, V_{th}]$ when $N \to \infty$, except in the extreme case when the input current at different time steps distributes extremely unevenly. So we can just assume that $V^+[N] \in [0, V_{th}]$. Furthermore, if we set a significantly smaller threshold V_{th} compared to the magnitude of $\hat{I}[N]$, the term $\frac{V^+[N]}{\tau}$ can be ignored. Then from Eq. (S3), we have

$$\lim_{N\to\infty} \hat{a}[N] \approx \text{clamp}\left(\lim_{N\to\infty} \frac{\hat{I}[N]}{\tau}, 0, \frac{V_{th}}{\Delta t}\right). \tag{S4}$$

That is ,we can approximate $\hat{a}[N]$ by $\text{clamp}\left(\frac{\hat{I}[N]}{\tau}, 0, \frac{V_{th}}{\Delta t}\right)$, with an approximation error bounded by $\frac{V_{th}}{\tau}$ when $N \to \infty$.

In summary, we derive Eq. (14) from Eq. (13) in the main content under following mild conditions:

1. The LIF neuron fires no or finite spikes as $N \to \infty$ when $\frac{\hat{I}[N]}{\tau} < 0$. And the LIF neuron does not fire only at a finite number of time steps as $N \to \infty$ when $\frac{\hat{I}[N]}{\tau} > \frac{V_{th}}{\Delta t}$.
2. $V^+[N] \in [0, V_{th}]$.

A.2. Derivation for Eq. (15)

In this subsection, we consider the IF model defined by Eqs. (3a) to (3c) and (4) and derive Eq. (15) in the main content under mild assumptions.

Combining Eqs. (4) and (3c), and taking the summation over $n = 1$ to N, we can get

$$V[N] - V[0] = \sum_{n=1}^{N} I[n] - V_{th} \sum_{n=1}^{N} s[n]. \tag{S5}$$

Define the scaled firing rate until the time step N as $a[N] = \frac{1}{N} \sum_{n=1}^{N} V_{th} s[n]$, and the average input current as $\bar{I}[N] = \frac{1}{N} \sum_{n=1}^{N} I[n]$. Dividing Eq. (S5) by N, we have

$$a[N] = \bar{I}[N] - \frac{V[N]}{N}. \tag{S6}$$

Using similar arguments appeared in Section A.1, we can get

$$\lim_{N\to\infty} a[N] = \lim_{N\to\infty} \text{clamp}\left(\bar{I}[N] - \frac{V^+[N]}{N}, 0, V_{th}\right), \tag{S7}$$

if the limit of right hand side exists. Here $V^+[N] = V[N] - V^-[N]$ and

$$V^-[N] = \min\left(\max\left(\sum_{n=1}^{N} I[n] - NV_{th}, 0\right), \sum_{n=1}^{N} I[n]\right). \tag{S8}$$

Similar to the LIF model, the "remaining" membrane potential $V^+[N]$ for the IF model is typically bounded in $[0, V_{th}]$ when $N \to \infty$, except in the extreme case. For example, consider $\bar{I}[N] > 0$ and the input current is non-zero only at the last $\frac{N}{2}$ time steps, then $V^+[N]$ will be inconsistently large and will be unbounded when $N \to \infty$. However, this extreme case will not happen in SNN computation for normal input data. Therefore, we assume $V^+[N] \in [0, V_{th}]$, and can get

$$\lim_{N\to\infty} a[N] = \text{clamp}\left(\lim_{N\to\infty} \bar{I}[N], 0, V_{th}\right). \tag{S9}$$

if the limit of $\bar{I}[N]$ exists.

In summary, we derive Eq. (15) in the main content under following mild conditions:

1. The IF neuron fires no or finite spikes as $N \to \infty$ when $\bar{I}[N] < 0$. And the IF neuron does not fire only at a finite number of time steps as $N \to \infty$ when $\bar{I}[N] > V_{th}$.

2. $V^+[N] \in [0, V_{th}]$.

B. Pseudocode of the Proposed DSR Method

We present the pseudocode of one iteration of SNN training with the DSR method in Algorithm 1 for better illustration.

Algorithm 1 One iteration of SNN training with the proposed DSR method.

Input: Time steps N; Network depth L; Network parameters $\mathbf{W}^L, \cdots, \mathbf{W}^L, V_{th}^1, \cdots, V_{th}^L$; Input data $\mathbf{x}$; Label $\mathbf{y}$; Other hyperparameters.
Output: Trained network parameters $\mathbf{W}^L, \cdots, \mathbf{W}^L$, $V_{th}^1, \cdots, V_{th}^L$.
Forward:
for $n = 1, 2, \cdots, N$ **do**
  for $i = 1, 2, \cdots, L$ **do**
    if the IF model is used **then**
      Calculate $\mathbf{s}^i[n]$ by Eq. (7);
    else if the LIF model is used **then**
      Calculate $\mathbf{s}^i[n]$ by Eq. (8);
    end if
    if $n = N$ **then**
      if the IF model is used **then**
        $\mathbf{o}^i = \frac{1}{N} \sum_{n=1}^{N} V_{th} \mathbf{s}[n]$;
      else if the LIF model is used **then**
        $\mathbf{o}^i = \frac{V_{th} \sum_{n=1}^{N} \lambda^{N-n} s[n]}{\sum_{n=1}^{N} \lambda^{N-n} \Delta t}$;
      end if
    end if
  end for
end for
Calculate the loss ℓ based on $\mathbf{o}^L$ and $\mathbf{y}$.
Backward:
Calculate $\frac{\partial \ell}{\partial \mathbf{o}^L}$;
for $i = L, L-1, \cdots, 1$ **do**
  Calculates $\frac{\partial \mathbf{o}^i}{\partial \mathbf{o}^{i-1}}$, $\frac{\partial \mathbf{o}^i}{\partial \mathbf{W}^i}$, and $\frac{\partial \mathbf{o}^i}{\partial V_{th}^i}$ by Eq. (17);
  $\frac{\partial \ell}{\partial \mathbf{W}^i} = \frac{\partial \ell}{\partial \mathbf{o}^i} \frac{\partial \mathbf{o}^i}{\partial \mathbf{W}^i}$;
  $\frac{\partial \ell}{\partial V_{th}^i} = \frac{\partial \ell}{\partial \mathbf{o}^i} \frac{\partial \mathbf{o}^i}{\partial V_{th}^i}$;
  if $i \neq 1$ **then**
    $\frac{\partial \ell}{\partial \mathbf{o}^{i-1}} = \frac{\partial \ell}{\partial \mathbf{o}^i} \frac{\partial \mathbf{o}^i}{\partial \mathbf{o}^{i-1}}$;
  end if
  Update $\mathbf{W}^i, V_{th}^i$ based on $\frac{\partial \ell}{\partial \mathbf{W}^i}$, $\frac{\partial \ell}{\partial V_{th}^i}$.
end for

C. Implementation Details

C.1. Dataset Description and Preprocessing

CIFAR-10 and CIFAR-100 The CIFAR-10 dataset [7] contains 60,000 32×32 color images in 10 different classes, which can be separated into 50,000 training samples and 10,000 testing samples. We apply data normalization to ensure that input images have zero mean and unit variance. We apply random cropping and horizontal flipping for data augmentation. The CIFAR-100 dataset [7] is similar to CIFAR-10 except that there are 100 classes of objects. We use the same data preprocessing as CIFAR-10. These two datasets are licensed under MIT.

ImageNet The ImageNet-1K dataset [1] spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. This dataset is licensed under Custom (non-commercial). We apply data normalization to ensure zero mean and unit variance for input images. Moreover, we apply random resized cropping and horizontal flipping for data augmentation.

DVS-CIFAR10 The DVS-CIFAR10 dataset [8] is a neuromophic dataset converted from CIFAR-10 using an event-based sensor. It contains 10,000 event-based images with resolution 128×128 pixels. The images are in 10 classes, with 1000 examples in each class. The dataset is licensed under CC BY 4.0. Since each spike train contains more than one million events, we split the events into 20 slices and integrate the events in each slice into one frame. More details about the transformation could be found in [3]. To conduct training and testing, we separate the whole data into 9000 training images and 1000 test images. Both the event-to-frame integrating and data separation are handled with the SpikingJelly [2] framework. We also reduce the spatial resolution from 128×128 to 48×48 and apply random cropping for data augmentation.

C.2. Batch Normalization

Batch Normalization (BN) [5] is a widely used technique in the deep learning community to stabilize signal propagation and accelerate training. In this paper, BN is adopted in the network architectures. However, since the input data for SNNs have an additional time dimension when compared to input image data for ANNs, we need to make the BN components suitable for SNNs.

In this paper, we combine the time dimension and the batch dimension into one and then conduct BN. In detail, consider a batch of temporal data $\mathbf{x} \in \mathbb{R}^{B\times N}$ with batch size B and temporal dimension N such that $\mathbf{x} = (\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(B)})$, and $\mathbf{x}^{(i)} \in \mathbb{R}^N$ for $i = 1, 2, \cdots, N$. Then define μ and σ^2 to be the mean and variance of the reshaped data $(x^{(1)}[1], \cdots, x^{(1)}[N], \cdots, x^{(B)}[1], \cdots, x^{(B)}[N])$. With the defined μ and σ^2, BN transforms the original data $\mathbf{x}^{(i)}$ to $\hat{\mathbf{x}}^{(i)}$ as

$$\hat{\mathbf{x}}^{(i)} = \gamma \frac{\mathbf{x}^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{S10}$$

where γ and β are learnable parameters, and ϵ is a small positive number to guarantee valid division.

C.3. Network Architectures

We use the pre-activation ResNet-18 [4] network architecture to conduct experiments on CIFAR-10, CIFAR-100, and ImageNet. To make the network architecture implementable on neuromorphic chips, we add spiking neurons after pooling operations and the last fully connected classifier. Furthermore, we replace all max pooling with average pooling. To stabilize the (weighted) firing rates of the output layer, we also introduce an additional BN operation between the last fully connected classifier and the last spiking neuron layer. The network contains four groups of basic block [4] structures, with channel sizes 64, 128, 256, and 512, respectively.

To test the effectiveness of the proposed DSR method with deeper networks structures, we conduct experiments with the pre-activation ResNet with 20, 32, 44, 56, 110 layers, whose architectures are shown in Tab. S1. We also add additional spiking neuron layers and BN layers like what we do for the PreAct-ResNet-18 structure.

We use the VGG-11 [11] network architecture to conduct experiments on DVS-CIFAR10. To enhance generalization capacity of the network, we further add dropout [12] layers after the spiking neurons, and we set the probability of zeroing elements to be 0.1. We only keep one fully connected layer to reduce the number of neurons.

C.4. Training Hyperparameters

First, we consider hyperparameters about the IF model. We set the initial threshold for each layer $V_{th}^i = 6$ and restrict it to be no less than 0.01 during training. We set α in Eq. (20) to be 0.5.

Then we consider hyperparameters about the LIF model. We fix the time constant τ^i for the i^{th} layer to be 1 for each i. The setting for initial V_{th}^i, lower bound for V_{th}^i, Δt, and α change with the number of time steps, as shown in Tab. S2.

Table S2. Hyperparameters for the LIF model given different number of time steps. "LB for V_{th}^i" means lower bound for V_{th}^i.

Time Steps	initial V_{th}^i	LB for V_{th}^i	Δt	α
20	0.3	0.0005	0.05	0.3
15	0.3	0.0005	0.05	0.4
10	0.3	0.0005	0.05	0.4
5	0.6	0.001	0.1	0.5

Next, we consider hyperparameters about the optimization. We use cosine annealing [9] as the learning rate schedule for all datasets. Other hyperparameters can be found in Tab. S3. Also note that we change the initial learning rate from 0.1 to 0.05 when using 5 time steps.

Table S3. Hyperparameters about Optimization for training CIFAR-10, CIFAR-100, ImageNet, and DVS-CIFAR10.

Dataset	Optimizer	Epoch	lr	Batchsize
CIFAR-10	SGD [10]	200	0.1	128
CIFAR-100	SGD [10]	200	0.1	128
ImageNet	Adam [6]	90	0.001	144
DVS-CIFAR10	SGD [10]	300	0.05	128

Table S1. Network architectures for PreAct-ResNet-20, PreAct-ResNet-32, PreAct-ResNet-44, PreAct-ResNet-56, PreAct-ResNet-110.

20-layers	32-layers	44-layers	56-layers	110-layers
conv (3×3,16)				
$\begin{pmatrix} 3\times3, 16 \\ 3\times3, 16 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3, 16 \\ 3\times3, 16 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3, 16 \\ 3\times3, 16 \end{pmatrix} \times 7$	$\begin{pmatrix} 3\times3, 16 \\ 3\times3, 16 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3, 16 \\ 3\times3, 16 \end{pmatrix} \times 18$
$\begin{pmatrix} 3\times3, 32 \\ 3\times3, 32 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3, 32 \\ 3\times3, 32 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3, 32 \\ 3\times3, 32 \end{pmatrix} \times 7$	$\begin{pmatrix} 3\times3, 32 \\ 3\times3, 32 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3, 32 \\ 3\times3, 32 \end{pmatrix} \times 18$
$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 3$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 5$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 7$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 9$	$\begin{pmatrix} 3\times3, 64 \\ 3\times3, 64 \end{pmatrix} \times 18$
average pool, 10-d fc				

D. Firing Sparsity

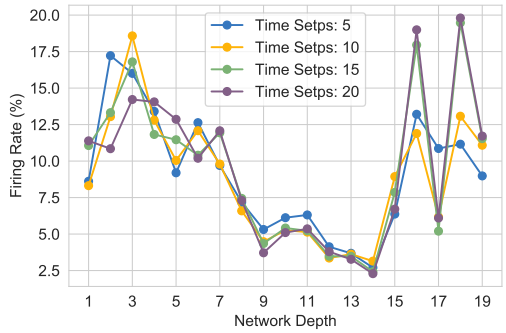
To achieve low energy consumption on neuromorphic hardware, the number of spikes generated by an SNN should be small. Then the firing rate is an important quantity to measure the energy efficiency of SNNs. We calculate the average firing rates of the trained SNNs on CIFAR-10, as shown in Fig. S1. The results show that the firing rates of all layers are below 20%, and many layers have firing rates of no more than 5%. Regardless of the layer of neurons, the total firing rates are between 7.5% and 9.5% for different number of time steps and both neuron models. Since the firing rate does not increase as the number of time steps decreases, the proposed method can achieve satisfactory performance with both low latency and high firing sparsity.

E. Weight Quantization

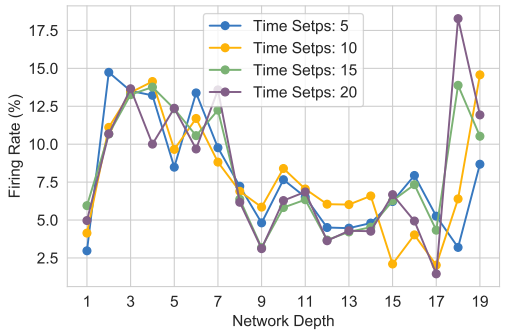
In our experiments, the network weights are 32-bit. However, we can also adopt low-bit weights when implementing our method on neuromorphic hardware, by combining existing quantization algorithms. The weights in neuromorphic hardware are generally 8-bit. So we simply quantize the weights of our trained SNNs to 8 bits and even 4 bits using the straight-through estimation method, and the results on CIFAR-10 are shown in Tab. S4.

Table S4. Performances on CIFAR-10 with network weights of different precisions. The PreAct-ResNet-18 architecture with 20 time steps is used.

Neural Model	32 bits	8 bits	4 bits
IF	95.38%	95.45%	95.31%
LIF	95.63%	95.65%	95.39%



(a) Firing rates for the IF model.



(b) Firing rates for the LIF model.

Figure S1. Firing rates of the trained SNNs by the proposed DSR method. The SNNs are trained on CIFAR-10 with the PreAct-ResNet-18 structure and different number of time steps. Firing rates of different spiking neuron layers are shown.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009. 3

[2] Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. Spikingjelly. https://github.com/fangwei123456/spikingjelly, 2020. 3

[3] 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. 3

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016. 3

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 3

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 3

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2

[8] 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. 3

[9] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 3

[10] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 3

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2014. 3

[12] 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. 3