Direct Training for Spiking Neural Networks: Faster, Larger, Better

Yujie Wu [1], **Lei Deng** [2], **Guoqi Li** [1], **Jun Zhu** [3†], **Yuan Xie** [2] **and Luping Shi** [1†]

[1]Center for Brain-Inspired Computing Research, Department of Precision Instrument, Tsinghua University
[2] Department of Electrical and Computer Engineering, University of California, Santa Barbara
[3] Department of Computer Science and Technology, Institute for AI, THBI Lab, Tsinghua University
[†]Corresponding: lpshi@tsinghua.edu.cn; dcszj@mail.tsinghua.edu.cn

Abstract

Spiking neural networks (SNNs) that enables energy efficient implementation on emerging neuromorphic hardware are gaining more attention. Yet now, SNNs have not shown competitive performance compared with artificial neural networks (ANNs), due to the lack of effective learning algorithms and efficient programming frameworks. We address this issue from two aspects: (1) We propose a neuron normalization technique to adjust the neural selectivity and develop a direct learning algorithm for deep SNNs. (2) Via narrowing the rate coding window and converting the leaky integrate-and-fire (LIF) model into an explicitly iterative version, we present a Pytorch-based implementation method towards the training of large-scale SNNs. In this way, we are able to train deep SNNs with tens of times speedup. As a result, we achieve significantly better accuracy than the reported works on neuromorphic datasets (N-MNIST and DVS-CIFAR10), and comparable accuracy as existing ANNs and pre-trained SNNs on non-spiking datasets (CIFAR10). To our best knowledge, this is the first work that demonstrates direct training of deep SNNs with high performance on CIFAR10, and the efficient implementation provides a new way to explore the potential of SNNs.

Introduction

Spiking neural networks, a sub-category of brain-inspired computing models, use spatio-temporal dynamics to mimic neural behaviors and binary spike signals to communicate between units. Benefit from the event-driven processing paradigm (computation occurs only when the unit receives spike event), SNNs can be efficiently implemented on specialized neuromorphic hardware for power-efficient processing, such as SpiNNaker (Khan et al. 2008), TrueNorth (Merolla et al. 2014), and Loihi (Davies et al. 2018).

As well known, the powerful error backpropagation (BP) algorithm and larger and larger model size enabled by ANN-oriented programming frameworks (e.g. Tensorflow, Pytorch) has boosted the wide applications of ANNs in recent years. However, the rich spatio-temporal dynamics and event-driven paradigm make SNNs much different from conventional ANNs. Till now SNNs have not yet demonstrated comparable performance to ANNs due to the lack

of effective learning algorithms and efficient programming frameworks, which greatly limit the network scale and application spectrum (Tavanaei et al. 2018).

In terms of learning algorithms, three challenges exist for training large-scale SNNs. First, the complex neural dynamics in both spatial and temporal domains make BP obscure. Specifically, the neural activities not only propagate layer by layer in spatial domain, but also affect the states along the temporal direction, which is more complicated than typical ANNs. Second, the event-driven spiking activity is discrete and non-differentiable, which impedes the BP implementation based on gradient descent. Third, SNNs are more sensitive to parameter configuration because of binary spike representation. Especially in the training phase, we should simultaneously ensure timely response for presynaptic stimulus and avoid too many spikes that will probably degrade the neuronal selectivity. Although previous work has proposed many techniques to adjust firing rate, such as model-based normalization (Diehl et al. 2015) and spike-based normalization (Sengupta et al. 2018), all of them are specialized for ANNs-to-SNNs conversion learning, not for the direct training of SNNs as considered in this paper.

In terms of programming frameworks, we lack suitable platforms to support the training of deep SNNs. Although there exist several platforms serving for simulating biological features of SNNs with varied abstraction levels (Brette et al. 2007; Carnevale and Hines 2006; Hazan et al. 2018), little work is designed for training deep SNNs. Researchers have to build application-oriented models from scratch and the training speed is usually slow. On the other hand, emerging ANN-oriented frameworks can provide much better efficiency, especially for large models, hence a natural idea is to map the SNN training onto these frameworks. However, these frameworks are designed for aforementioned ANN algorithms so that they cannot be directly applied to SNNs because of the neural dynamic of spiking neurons.

In this paper, we propose a full-stack solution towards faster, larger, and better SNNs from both algorithm and programming aspects. We draw inspirations from the recent work (Wu et al. 2018), which proposes the spatio-temporal back propagation (STBP) method for the direct training of SNNs, and significantly extend it to much deeper structure, larger dataset, and better performance. We first propose the NeuNorm method to balance neural selectivity and increase

the performance. Then we improve the rate coding to fasten the convergence and convert the LIF model into an explicitly iterative version to make it compatible with a machine learning framework (Pytorch). As a results, compared to the running time on Matlab, we achieve tens of times speedup that enables the direct training of deep SNNs. The best accuracy on neuromorphic datasets (N-MNIST and DVS-CIFAR10) and comparable accuracy as existing ANNs and pre-trained SNNs on non-spiking datasets (CIFAR10) are demonstrated. This work enables the exploration of direct training of high-performance SNNs and facilitates the SNN applications via compatible programming within the widely used machine learning (ML) framework.

Related work

We aims to direct training of deep SNNs. To this end, we mainly make improvements on two aspects: (1) learning algorithm design; (2) training acceleration optimization. We chiefly overview the related works in recent years.

Learning algorithm for deep SNNs. There exist three ways for SNN learning: i) unsupervised learning such as spike timing dependent plasticity (STDP); ii) indirect supervised learning such as ANNs-to-SNNs conversion; iii) direct supervised learning such as gradient descent-based back propagation. However, by far most of them limited to very shallow structure (network layer less than 4) or toy small dataset (e.g. MNIST, Iris), and little work points to direct training deep SNNs due to their own challenges.

STDP is biologically plausible but the lack of global information hinders the convergence of large models, especially on complex datasets (Timothe and Thorpe 2007; Diehl and Matthew 2015; Tavanaei and Maida 2017).

The ANNs-to-SNNs conversion (Diehl et al. 2015; Cao, Chen, and Khosla 2015; Sengupta et al. 2018; Hu et al. 2018) is currently the most successful method to model large-scale SNNs. They first train non-spiking ANNs and convert it into a spiking version. However, this indirect training help little on revealing how SNNs learn since it only implements the inference phase in SNN format (the training is in ANN format). Besides, it adds many constraints onto the pre-trained ANN models, such as no bias term, only average pooling, only ReLU activation function, etc. With the network deepens, such SNNs have to run unacceptable simulation time (100-1000 time steps) to obtain good performance.

The direct supervised learning method trains SNNs without conversion. They are mainly based on the conventional gradient descent. Different from the previous spatial back-propagation (Lee, Delbruck, and Pfeiffer 2016; Jin, Li, and Zhang 2018), (Wu et al. 2018) proposed the first backpropagation in both spatial and temporal domains to direct train SNNs, which achieved state-of-the-art accuracy on MNIST and N-MNIST datasets. However, the learning algorithm is not optimized well and the slow simulation hinders the exploration onto deeper structures.

Normalization. Many normalization techniques have been proposed to improve the convergence (Ioffe and Szegedy 2015; Wu and He 2018). Although these methods have achieved great success in ANNs, they are not suitable for SNNs due to the complex neural dynamics and binary spiking representation of SNNs. Furthermore, in terms of hardware implementation, batch-based normalization techniques essentially incur lateral operations across different stimuluses which are not compatible with existing neuromorphic platforms (Merolla et al. 2014; Davies et al. 2018). Recently, several normalization methods (e.g. model-based normalization (Diehl et al. 2015), data-based normalization (Diehl et al. 2015), spike-based normalization (Sengupta et al. 2018)) have been proposed to improve SNN performance. However, such methods are specifically designed for the indirect training with ANNs-to-SNNs conversion, which didn't show convincing effectiveness for direct training of SNNs targeted by this work.

SNN programming frameworks. There exist several programming frameworks for SNN modeling, but they have different aims. NEURON (Carnevale and Hines 2006) and Genesis (Bower and Beeman 1998) mainly focus on the biological realistic simulation from neuron functionality to synapse reaction, which are more beneficial for the neuroscience community. BRIAN2 (Goodman and Brette 2009) and NEST (Gewaltig and Diesmann 2007) target the simulation of larger scale SNNs with many biological features, but not designed for the direct supervised learning for high performance discussed in this work. BindsNET (Hazan et al. 2018) is the first reported framework towards combining SNNs and practical applications. However, to our knowledge, the support for direct training of deep SNNs is still under development. Furthermore, according to the statistics from ModelDB [1], in most cases researchers even simply program in general-purpose language, such as C/C++ and Matlab, for better flexibility. Besides the different aims, programming from scratch on these frameworks is user unfriendly and the tremendous execution time impedes the development of large-scale SNNs. Instead of developing a new framework, we provide a new solution to establish SNNs by virtue of mature ML frameworks which have demonstrated easy-to-use interface and fast running speed.

Approach

In this section, we first convert the LIF neuron model into an easy-to-program version with the format of explicit iteration. Then we propose the NeuNorm method to adjust the neuronal selectivity for improving model performance. Furthermore, we optimize the rate coding scheme from encoding and decoding aspects for faster response. Finally, we describe the whole training and provide the pseudo codes for Pytorch.

Explicitly iterative LIF model

LIF model is commonly used to describe the behavior of neuronal activities, including the update of membrane potential and spike firing. Specifically, it is governed by

$$\tau \frac{du}{dt} = -u + I, \quad u < V_{th} \tag{1}$$

$$\text{fire a spike \& } u = u_{reset}, \quad u \geq V_{th} \tag{2}$$

---

[1]ModelDB is an open website for storing and sharing computational neuroscience models.

where u is the membrane potential, τ is a time constant, I is pre-synaptic inputs, and V_{th} is a given fire threshold. Eq. (1)-(2) describe the behaviors of spiking neuron in a way of updating-firing-resetting mechanism (see Fig. 1a-b). When the membrane potential reaches a given threshold, the neuron will fire a spike and u is reset to u_{reset}; otherwise, the neuron receives pre-synapse stimulus I and updates its membrane potential according to Eq. (1).
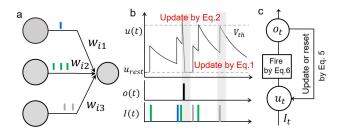

Figure 1: **Illustration of iterative LIF**. Spike communication between neurons; (b) The update of membrane potential according to Eq. (1)-Eq. (2); (c) Iterative LIF described by Eq. (5)-Eq. (6).

The above differential expressions described in continuous domain are widely used for biological simulations, but it is too implicit for the implementations on mainstream ML frameworks (e.g. Tensorflow, Pytorch) since the embedded automatic differentiation mechanism executes codes in a discrete and sequential way (Paszke et al. 2017). Therefore, it motivates us to convert Eq. (1)-(2) into an explicitly iterative version for ensuring computational tractability. To this end, we first use Euler method to solve the first-order differential equation of Eq. (1), and obtain an iterative expression

$$u^{t+1} = (1 - \frac{dt}{\tau})u^t + \frac{dt}{\tau}I. \quad (3)$$

We denote the factor $(1 - \frac{dt}{\tau})$ as a decay factor $k_{\tau 1}$ and expand the pre-synaptic input I to a linearing summation $\sum_j W_j o(j)$. The subscript j indicates the index of pre-synapse and $o(j)$ denotes the corresponding pre-synaptic spike which is binary (0 or 1). By incorporating the scaling effect of $\frac{dt}{\tau}$ into synapse weights W, we have

$$u^{t+1} = k_{\tau 1}u^t + \sum_j W_j o(j). \quad (4)$$

Next we add the firing-and-resetting mechanism to Eq. (4). By assuming $u_{reset} = 0$ as usual, we get the final update equations as below

$$u^{t+1,n+1}(i) = k_{\tau 1}u^{t,n+1}(i)(1 - o^{t,n+1}(i)) + \sum_{j=1}^{l(n)} w_{ij}^n o^{t+1,n}(j), \quad (5)$$

$$o^{t+1,n+1}(i) = f(u^{t+1,n+1}(i) - V_{th}), \quad (6)$$

where n and $l(n)$ denote the n-th layer and its neuron number, respectively, w_{ij} is the synaptic weight from the j-th neuron in pre-layer (n) to the i-th neuron in the post-layer $(n+1)$. $f(\cdot)$ is the step function, which satisfies $f(x) = 0$

when $x < 0$, otherwise $f(x) = 1$. Eq (5)-(6) reveal that firing activities of $o^{t,n+1}$ will affect the next state $o^{t+1,n+1}$ via the updating-firing-resetting mechanism (see Fig. 1c). If the neuron emits a spike at time step t, the membrane potential at $t + 1$ step will clear its decay component $k_{\tau 1}u_t$ via the term $(1 - o_j^{t,n+1})$, and vice versa.

Through Eq. (5)-(6), we convert the implicit Eq. (1)-(2) into an explicitly version, which is easier to implement on ML framework. We give a concise pseudo code based on Pytorch for an explicitly iterative LIF model in Algorithm 1.

Algorithm 1 State update for an explicitly iterative LIF neuron at time step $t + 1$ in the $(n + 1)$-th layer.

Require: Previous potential $u^{t,n+1}$ and spike output $o^{t,n+1}$, current spike input $o^{t+1,n}$, and weight vector W
Ensure: Next potential $u^{t+1,n+1}$ and spike output $o^{t+1,n+1}$
    Function StateUpdate(W,$u^{t,n+1}$, $o^{t,n+1}$, $o^{t+1,n}$)
1:   $u^{t+1,n+1} = k_{\tau 1}u^{t,n+1}(1 - o^{t,n+1}) + Wo^{t+1,n}$;
2:   $o^{t+1,n+1} = f(u^{t+1,n+1} - V_{th})$
3: **return** $u^{t+1,n+1}, o^{t+1,n+1}$

Neuron normalization (NeuNorm)

Considering the signal communication between two convolutional layers, the neuron at the location (i, j) of the f-th feature map (FM) in the n-th layer receives convolutional inputs I, and updates its membrane potential by

$$u_f^{t+1,n+1}(i,j) = k_{\tau 1}u_f^{t,n+1}(i,j)(1 - o_f^{t,n+1}(i,j)) + I_f^{t+1,n+1}(i,j),$$

$$(7)$$

$$I_f^{t+1,n+1}(i,j) = \sum_c W_{c,f}^n \circledast o_c^{t+1,n}(R(i,j)), \quad (8)$$

where $W_{c,f}^n$ denotes the weight kernel between the c-th FM in layer n and the f-th FM in layer $n + 1$, $\circledast$ denotes the convolution operation, and $R(i, j)$ denotes the local receptive filed of location (i, j).

An inevitable problem for training SNNs is to balance the whole firing rate because of the binary spike communication (Diehl et al. 2015; Wu et al. 2018). That is to say, we need to ensure timely and selective response to pre-synaptic stimulus but avoid too many spikes that probably harm the effective information representation. In this sense, it requires that the strength of stimulus maintains in a relatively stable range to avoid activity vanishing or explosion as network deepens.

Observing that neurons respond to stimulus from different FMs, it motivates us to propose an auxiliary neuron method for normalizing the input strength at the same spatial locations of different FMs (see Fig. 2). The update of auxiliary neuron status x is described by

$$x^{t+1,n}(i,j) = k_{\tau 2}x^{t,n}(i,j) + \frac{v}{F}\sum_f o_f^{t+1,n}(i,j), \quad (9)$$

where $k_{\tau 2}$ denotes the decay factor, v denotes the constant scaling factor, and F denotes the number of FMs in the n-th layer. In this way, Eq.(9) calculates the average response to the input firing rate with momentum term $k_{\tau 2}x^{t,n}(i,j)$. For simplicity we set $k_{\tau 2} + vF = 1$.
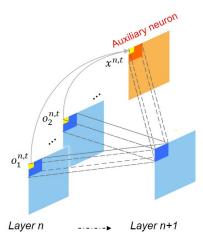
Figure 2: **Illustration of NeuNorm**. Blue squares represent the normal neuronal FMs, and orange squares represents the auxiliary neuronal FMs. The auxiliary neurons receive lateral inputs (solid lines), and fire signals (dotted lines) to control the strength of stimulus emitted to the next layer.

Next we suppose that the auxiliary neuron receives the lateral inputs from the same layer, and transmits signals to control the strength of stimulus emitted to the next layer through trainable weights U_c^n, which has the same size as the FM. Hence, the inputs I of neurons in the next layer can be modified by

$$I_f^{t+1,n+1}(i,j) = \sum_c W_{c,f}^n \circledast (o_c^{t+1,n}(R(i,j)) - ...$$
$$U_c^n(R(i,j)) \cdot x^{t+1,n}(R(i,j))). \quad (10)$$

Inferring from Eq. (10), NeuNorm method essentially normalize the neuronal activity by using the input statistics (moving average firing rate). The basic operation is similar with the zero-mean operation in batch normalization. But NeuNorm has different purposes and different data processing ways (normalizing data along the channel dimension rather than the batch dimension). This difference brings several benefits which may be more suitable for SNNs. Firstly, NeuNorm is compatible with neuron model (LIF) without additional operations and friendly for neuromorphic implementation. Besides that, NeuNorm is more bio-plausible. Indeed, the biological visual pathways are not independent. The response of retina cells for a particular location in an image is normalized across adjacent cell responses in the same layer (Carandini and Heeger 2012; Mante, Bonin, and Carandini 2008). And inspired by these founding, many similar techniques for exploiting channel features have been successfully applied to visual recognition, such as group normalization (GN) (Wu and He 2018), DOG (Daral 2005) and HOG (Daral 2005). For example, GN divides FMs of the same layer into several groups and normalizes the features within each group also along channel dimension.

Encoding and decoding schemes

To handle various stimulus patterns, SNNs often use abundant coding methods to process the input stimulus. For visual recognition tasks, a popular coding scheme is rate coding. On the input side, the real-valued images are converted into a spike train whose fire rate is proportional to the pixel intensity. The spike sampling is probabilistic, such as following a Bernoulli distribution or a Poisson distribution. On the output side, it counts the firing rate of each neuron in the last layer over given time windows to determine network output. However, the conventional rate coding suffers from long simulation time to reach good performance. To solve this problem, we take a simpler coding scheme to accelerate the simulation without compromising the performance.

Encoding. One requirement for long simulation time is to reduce the sampling error when converting real-value inputs to spike signals. Specifically, given the time window T, a neuron can represent information by $T + 1$ levels of firing rate, i.e. $\{0, \frac{1}{T}, ..., 1\}$ (normalized). Obviously, the rate coding requires sufficient long window to achieve satisfactory precision. To address this issue, we assign the first layer as an encoding layer and allow it to receive both spike and non-spike signals (compatible with various datasets). In other words, neurons in the encoding layer can process the even-driven spike train from neuromorphic dataset naturally, and can also convert real-valued signals from non-spiking dataset into spike train with enough precision. In this way, the precision is remained to a great extent without depending much on the simulation time.

Decoding. Another requirement for long time window is the representation precision of network output. To alleviate it, we adopt a voting strategy (Diehl and Matthew 2015) to decode the network output. We configure the last layer as a voting layer consisting of several neuron populations and each output class is represented by one population. In this way, the burden of representation precision of each neuron in the temporal domain (firing rate in a given time windows) is transferred much to the spatial domain (neuron group coding). Thus, the requirement for long simulation time is significantly reduced. For initialization, we randomly assign a label to each neuron; while during training, the classification result is determined by counting the voting response of all the populations.

In a nutshell, we reduce the demand for long-term window from above two aspects, which in some sense extends the representation capability of SNNs in both input and output sides. We found similar coding scheme is also leveraged by previous work on ANN compression (Tang, Hua, and Wang 2017; Hubara, Soudry, and Ran 2016). It implies that, regardless of the internal lower precision, maintaining the first and last layers in higher precision are important for the convergence and performance.

Overall training implementation

We define a loss function L measuring the mean square error between the averaged voting results and label vector Y

within a given time window T

$$L = \parallel Y - \frac{1}{T}\sum_{t=1}^{T} Mo^{t,N} \parallel_2^2, \qquad (11)$$

where $o^{t,N}$ denotes the voting vector of the last layer N at time step t. M denotes the constant voting matrix connecting each voting neuron to a specific category.

From the explicitly iterative LIF model, we can see that the spike signals not only propagate through the layer-by-layer spatial domain, but also affect the neuronal states through the temporal domain. Thus, gradient-based training should consider both the derivatives in these two domains. Based on this analysis, we integrate our modified LIF model, coding scheme, and proposed NeuNorm into the STBP method (Wu et al. 2018) to train our network. When calculating the derivative of loss function L with respect to u and o in the n-th layer at time step t, the STBP propagates the gradients $\frac{\partial L}{\partial o_i^{t,n+1}}$ from the $(n+1)$-th layer and $\frac{\partial L}{\partial o_i^{t+1,n}}$ from time step $t+1$ as follows

$$\frac{\partial L}{\partial o_i^{t,n}} = \sum_{j=1}^{l(n+1)} \frac{\partial L}{\partial o_j^{t,n+1}} \frac{\partial o_j^{t,n+1}}{\partial o_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial o_i^{t,n}}, \quad (12)$$

$$\frac{\partial L}{\partial u_i^{t,n}} = \frac{\partial L}{\partial o_i^{t,n}} \frac{\partial o_i^{t,n}}{\partial u_i^{t,n}} + \frac{\partial L}{\partial o_i^{t+1,n}} \frac{\partial o_i^{t+1,n}}{\partial u_i^{t,n}}. \qquad (13)$$

A critical problem in training SNNs is the non-differentiable property of the binary spike activities. To make its gradient available, we take the rectangular function $h(u)$ to approximate the derivative of spike activity (Wu et al. 2018). It yields

$$h(u) = \frac{1}{a} sign(|u - V_{th}| < \frac{a}{2}), \qquad (14)$$

where the width parameter a determines the shape of $h(u)$. Theoretically, it can be easily proven that Eq. (14) satisfies

$$\lim_{a \to 0^+} h(u) = \frac{df}{du}. \qquad (15)$$

Based on above methods, we also give a pseudo code for implementing the overall training of proposed SNNs in Pytorch, as shown in Algorithm 2.

Experiment

We test the proposed model and learning algorithm on both neuromorphic datasets (N-MNIST and DVS-CIFAR10) and non-spiking datasets (CIFAR10) from two aspects: (1) training acceleration; (2) application accuracy. The dataset introduction, pre-processing, training detail, and parameter configuration are summarized in Appendix.

Network structure

Tab. 1 and Tab. 2 provide the network structures for acceleration analysis and accuracy evaluation, respectively. The structure illustrations of what we call AlexNet and CIFAR-Net are also shown in Appendix, which are for fair comparison to pre-trained SNN works with similar structure and to demonstrate comparable performance with ANNs, respectively. It is worth emphasizing that previous work on direct

---

Algorithm 2 Training codes for one iteration.

Require: Network inputs $\{X^t\}_t^T$, class label Y, parameters and states of convolutional layers ($\{W^l, U^l, u^{0,l}, o^{0,l}\}_{l=1}^{N_1}$) and fully-connected layers ($\{W^l, u^{0,l}, o^{0,l}\}_{l=1}^{N_2}$), simulation windows T, voting matrix M // Map each voting neuron to label.
Ensure: Update network parameters.

    Forward (inference):
1: **for** $t = 1$ to T **do**
2:     $o^{t,1} \leftarrow$ EncodingLayer(X^t)
3:     **for** $l = 2$ to N_1 **do**
4:        $x^{t,l-1} \leftarrow$ AuxiliaryUpdate($x^{t-1,l-1}, o^{t,l-1}$)   // Eq. (9).
5:        $(u^{t,l}, o^{t,l}) \leftarrow$ StateUpdate($W^{l-1}, u^{t-1,l}, o^{t-1,l}, o^{t,l-1}, U^{l-1}, x^{t,l-1}$)   // Eq. (6)-(7), and (9)-(10)
6:     **end for**
7:     **for** $l = 1$ to N_2 **do**
8:        $(u^{t,l}, o^{t,l}) \leftarrow$ StateUpdate($W^{l-1}, u^{t-1,l}, o^{t-1,l}, o^{t,l-1}$) // Eq. (5)-(6)
9:     **end for**
10: **end for**

    Loss:
11: $L \leftarrow$ ComputeLoss(Y, M, o^{t,N_2})   // Eq. (11).

    Backward:
12: Gradient initialization: $\frac{\partial L}{\partial o^{T+1,*}} = 0$.
13: **for** $t = T$ to 1 **do**
14:     $\frac{\partial L}{\partial o^{t,N_2}} \leftarrow$ LossGradient($L, M, \frac{\partial L}{\partial o^{t+1,N_2}}$)
      // Eq. (5)-(6), and (11)-(13).
15:     **for** $l = N_2 - 1$ to 1 **do**
16:        $(\frac{\partial L}{\partial o^{t,l}}, \frac{\partial L}{\partial u^{t,l}}, \frac{\partial L}{\partial W^l}) \leftarrow$ BackwardGradient($\frac{\partial L}{\partial o^{t,l+1}}, \frac{\partial L}{\partial o^{t+1,l}}, W^l$)   // Eq. (5)-(6), and (12)-(13).
17:     **end for**
18:     **for** $l = N_1$ to 2 **do**
19:        $(\frac{\partial L}{\partial o^{t,l}}, \frac{\partial L}{\partial u^{t,l}}, \frac{\partial L}{\partial W^{l-1}}, \frac{\partial L}{\partial U^{l-1}}) \leftarrow$ BackwardGradient($\frac{\partial L}{\partial o^{t,l+1}}, \frac{\partial L}{\partial o^{t+1,l}}, W^{l-1}, U^{l-1}$)
      // Eq. (6)-(7), (9)-(10), and (12)-(13).
20:     **end for**
21: **end for**

22: **Update parameters based on gradients**.

Note: All the parameters and states with layer index l in the N_1 or N_2 loop belong to the convolutional or fully-connected layers, respectively. For clarity, we just use the symbol of l.

---

training of SNNs demonstrated only shallow structures (usually 2-4 layers), while our work for the first time can implement a direct and effective learning for larger-scale SNNs (e.g. 8 layers).

Training acceleration

Runtime. Since Matlab is a high-level language widely used in SNN community, we adopt it for the comparison with our Pytorch implementation. For fairness, we made several configuration restrictions, such as software version, parameter setting, etc. More details can be found in Appendix.

Fig. 3 shows the comparisons about average runtime per epoch, where batch size of 20 is used for simulation. Py-

Table 1: Network structures used for training acceleration.

	Neuromorphic Dataset
Small	128C3(Encoding)-AP2-128C3-AP2-512FC-Voting
Middle	128C3(Encoding)-128C3-AP2-256-AP2-1024FC-Voting
Large	128C3(Encoding)-128C3-AP2-384C3-384C3-AP2-1024FC-512FC-Voting
	Non-spiking Dataset
Small	128C3(Encoding)-AP2-256C3-AP2-256FC-Voting
Middle	128C3(Encoding)-AP2-256C3-512C3-AP2-512FC-Voting
Large	128C3(Encoding)-256C3-AP2-512C3-AP2-1024C3-512C3-1024FC-512FC-Voting

Table 2: Network structures used for accuracy evaluation.

	Neuromorphic Dataset
Our model	128C3(Encoding)-128C3-AP2-128C3-256C3-AP2-1024FC-Voting
	Non-spiking Dataset
AlexNet	96C3(Encoding)-256C3-AP2-384C3-AP2-384C3-256C3-1024FC-1024FC-Voting
CIFARNet	128C3(Encoding)-256C3-AP2-512C3-AP2-1024C3-512C3-1024FC-512FC-Voting

torch is able to provide tens of times acceleration on all three datasets. This improvement may be attributed to the specialized optimizations for the convolution operation in Pytorch. In contrast, currently these optimizations have not been well supported in most of existing SNN platforms. Hence building spiking model may benefit a lot from DL platform.
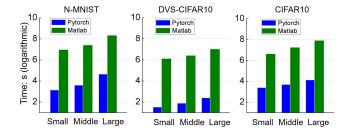

Figure 3: Average runtime per epoch.

Network scale. If without acceleration, it is difficult to extend most existing works to large scale. Fortunately, the fast implementation in Pytorch facilitates the construction of deep SNNs. This makes it possible to investigate the influence of network size on model performance. Although it has been widely studied in ANN field, it has yet to be demonstrated on SNNs. To this end, we compare the accuracy under different network sizes, shown in Fig. 4. With the size increases, SNNs show an apparent tendency of accuracy improvement, which is consistent with ANNs.

Simulation length. SNNs need enough simulation steps to mimic neural dynamics and encode information. Given simulation length T, it means that we need to repeat the inference process T times to count the firing rate. So the network computation cost can be denoted as $O(T)$. For deep SNNs, previous work usually requires 100 even 1000 steps to reach good performance (Sengupta et al. 2018), which brings huge computational cost. Fortunately, by using the
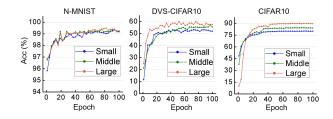

Figure 4: **Influence of network scale**. Three different network scales are defined in Tab.1.

proposed coding scheme in this work, the simulation length can be significantly reduced without much accuracy degradation. As shown in Fig. 5, although the longer the simulation windows leads to better results, our method just requires a small length $(4 - 8)$ to achieve satisfactory results. Notably, even if extremely taking one-step simulation, it can also achieve not bad performance with significantly faster response and lower energy, which promises the application scenarios with extreme restrictions on response time and energy consumption.
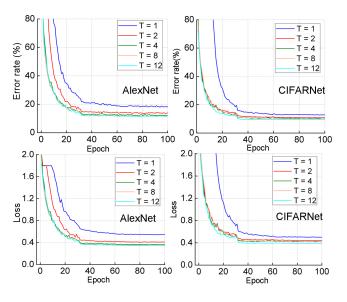

Figure 5: Influence of simulation length on CIFAR10.

Application accuracy

Neuromorphic datasets. Tab. 3 records the current state-of-the-art results on N-MNIST datasets. (Li 2018) proposes a technique to restore the N-MNIST dataset back to the static MNIST, and achieves 99.23% accuracy using non-spiking CNNs. Our model can naturally handle raw spike streams using direct training and achieves significantly better accuracy. Furthermore, by adding NeuNorm technique, the accuracy can be improved up to 99.53%.

Tab. 4 further compares the accuracy on DVS-CIFAR10 dataset. DVS-CIFAR10 is more challenging than N-MNIST due to the larger scale, and it is also more challenging than

Table 3: Comparison with existing results on N-MNIST.

Model	Method	Accuracy
(Neil, Pfeiffer, and Liu 2016)	LSTM	97.38%
(Lee, Delbruck, and Pfeiffer 2016)	Spiking NN	98.74%
(Wu et al. 2018)	Spiking NN	98.78%
(Jin, Li, and Zhang 2018)	Spiking NN	98.93%
(Neil and Liu 2016)	Non-spiking CNN	98.30%
(Li 2018)	Non-spiking CNN	99.23%
Our model	without NeuNorm	**99.44%**
Our model	with NeuNorm	**99.53%**

Table 4: Comparison with existing results on DVS-CIFAR10.

Model	Method	Accuracy
(Orchard et al. 2015a)	Random Forest	31.0%
(Lagorce et al. 2017)	HOTS	27.1%
(Sironi et al. 2018)	HAT	52.4%
(Sironi et al. 2018)	Gabor-SNN	24.5%
Our model	without NeuNorm	**58.1%**
Our model	with NeuNorm	**60.5%**

non-spiking CIFAR10 due to less samples and noisy environment (Dataset introduction in Appendix). Our model achieves the best performance with 60.5%. Moreover, experimental results indicate that NeuNorm can speed up the convergence. Without NeuNorm the required training epochs to get the best accuracy 58.1% is 157, while its reduced to 103 with NeuNorm.

Non-spiking dataset. Tab. 5 summarizes the results of existing state-of-the-art results and our CIFARNet on CIFAR10 dataset. Prior to our work, the best direct training method only achieves 75.42% accuracy. Our model is significantly better than this result, with 15% improvement (reaching 90.53%). We also make a comparison with non-spiking ANN model and other pre-trained works (ANNs-to-SNNs conversion) on the similar structure of AlexNet, wherein our direct training of SNNs with NeuNorm achieves slightly better accuracy.

Table 5: Comparison with existing state-of-the-art results on non-spiking CIFAR10.

Model	Method	Accuracy
(Panda and Roy 2016)	Spiking NN	75.42%
(Cao, Chen, and Khosla 2015)	Pre-trained SNN	77.43%
(Rueckauer et al. 2017)	Pre-trained SNN	90.8%
(Sengupta et al. 2018)	Pre-trained SNN	87.46%
Baseline	Non-spiking NN	**90.49%**
Our model	without NeuNorm	**89.83%**
Our model	with NeuNorm	**90.53%**

Table 6: Comparison with previous results using similar AlexNet struture on non-spiking CIFAR10.

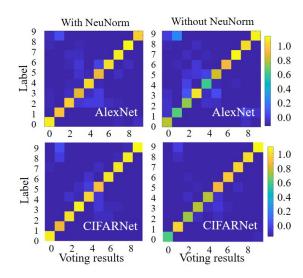| Model	Method	Accuracy
(Hunsberger and Eliasmith 2015)	Non-spiking NN	83.72%
(Hunsberger and Eliasmith 2015)	Pre-trained SNN	83.52%
(Sengupta et al. 2018)	Pre-trained SNN	83.54%
Our model	—	**85.24%**

Figure 6: **Confusion matrix of voting output with or without NeuNorm**. High values along the diagonal indicate correct recognition whereas high values anywhere else indicate confusion between two categories.

Furthermore, to visualize the differences of learning with or without NeuNorm, Fig. 6 shows the average confusion matrix of network voting results on CIFAR10 over 500 randomly selected images. Each location in the 10×10 tiles is determined by the voting output and the actual labels. High values along the diagonal indicate correct recognition whereas high values anywhere else indicate confusion between two categories. It shows that using NeuNorm brings a clearer contrast (i.e. higher values along the diagonal), which implies that NeuNorm enhances the differentiation degree among the 10 classes. Combining the results from Tab. 3-5, it also confirms the effectiveness of proposed NeuNorm for performance improvement.

Conclusion

In this paper, we present a direct training algorithm for deeper and larger SNNs with high performance. We propose the NeuNorm to effectively normalize the neuronal activities and improve the performance. Besides that, we optimize the rate coding from encoding aspect and decoding aspect, and convert the original continuous LIF model to an explicitly iterative version for friendly Pytorch implementation. Finally, through tens of times training accelerations and larger network scale, we achieve the best accuracy on neuromorphic datasets and comparable accuracy with ANNs on non-spiking datasets. To our best knowledge, this is the first time report such high performance with direct training on SNNs. The implementation on mainstream ML framework could facilitate the SNN development.

Acknowledgments

References

[Bower and Beeman 1998] Bower, J. M., and Beeman, D. 1998. *The Book of GENESIS*. Springer New York.

[Brette et al. 2007] Brette, R.; Rudolph, M.; Carnevale, T.; Hines, M.; Beeman, D.; Bower, J. M.; Diesmann, M.; Morrison, A.; Goodman, P. H.; and Harris, F. C. 2007. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience* 23(3):349–398.

[Cao, Chen, and Khosla 2015] Cao, Y.; Chen, Y.; and Khosla, D. 2015. *Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition*. Kluwer Academic Publishers.

[Carandini and Heeger 2012] Carandini, M., and Heeger, D. J. 2012. Normalization as a canonical neural computation. *Nature Reviews Neuroscience* 13(1):51–62.

[Carnevale and Hines 2006] Carnevale, N. T., and Hines, M. L. 2006. The neuron book. *Cambridge Univ Pr*.

[Daral 2005] Daral, N. 2005. Histograms of oriented gradients for human detection. *Proc Cvpr*.

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

[Diehl and Matthew 2015] Diehl, P. U., and Matthew, C. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity:. *Frontiers in Computational Neuroscience* 9:99.

[Diehl et al. 2015] Diehl, P. U.; Neil, D.; Binas, J.; Cook, M.; Liu, S. C.; and Pfeiffer, M. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *International Joint Conference on Neural Networks*, 1–8.

[Gewaltig and Diesmann 2007] Gewaltig, M. O., and Diesmann, M. 2007. Nest (neural simulation tool). *Scholarpedia* 2(4):1430.

[Goodman and Brette 2009] Goodman, D. F., and Brette, R. 2009. The brian simulator. *Frontiers in Neuroscience* 3(2):192.

[Hazan et al. 2018] Hazan, H.; Saunders, D. J.; Khan, H.; Sanghavi, D. T.; Siegelmann, H. T.; and Kozma, R. 2018. Bindsnet: A machine learning-oriented spiking neural networks library in python. *arXiv preprint arXiv:1806.01423*.

[Hu et al. 2018] Hu, Y.; Tang, H.; Wang, Y.; and Pan, G. 2018. Spiking deep residual network. *arXiv preprint arXiv:1805.01352*.

[Hubara, Soudry, and Ran 2016] Hubara, I.; Soudry, D.; and Ran, E. Y. 2016. Binarized neural networks. *arXiv preprint arXiv:1602.02505*.

[Hunsberger and Eliasmith 2015] Hunsberger, E., and Eliasmith, C. 2015. Spiking deep networks with lif neurons. *Computer Science*.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *JMLR* 448–456.

[Jin, Li, and Zhang 2018] Jin, Y.; Li, P.; and Zhang, W. 2018. Hybrid macro/micro level backpropagation for training deep spiking neural networks. *arXiv preprint arXiv:1805.07866*.

[Khan et al. 2008] Khan, M. M.; Lester, D. R.; Plana, L. A.; and Rast, A. 2008. Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor. In *IEEE International Joint Conference on Neural Networks*, 2849–2856.

[Lagorce et al. 2017] Lagorce, X.; Orchard, G.; Gallupi, F.; Shi, B. E.; and Benosman, R. 2017. Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP(99):1–1.

[Lee, Delbruck, and Pfeiffer 2016] Lee, J. H.; Delbruck, T.; and Pfeiffer, M. 2016. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience* 10.

[Li et al. 2017] Li, H.; Liu, H.; Ji, X.; Li, G.; and Shi, L. 2017. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience* 11.

[Li 2018] Li, L. I. . Y. C. . H. 2018. Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv preprint arXiv:1807.01013* 1–23.

[Mante, Bonin, and Carandini 2008] Mante, V.; Bonin, V.; and Carandini, M. 2008. Functional mechanisms shaping lateral geniculate responses to artificial and natural stimuli. *Neuron* 58(4):625–638.

[Merolla et al. 2014] Merolla, P. A.; Arthur, J. V.; Alvarezicaza, R.; Cassidy, A. S.; Sawada, J.; Akopyan, F.; Jackson, B. L.; Imam, N.; Guo, C.; and Nakamura, Y. 2014. Artificial brains. a million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345(6197):668–673.

[Neil and Liu 2016] Neil, D., and Liu, S. C. 2016. Effective sensor fusion with event-based sensors and deep network architectures. In *IEEE International Symposium on Circuits and Systems*.

[Neil, Pfeiffer, and Liu 2016] Neil, D.; Pfeiffer, M.; and Liu, S. C. 2016. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *arXiv preprint arXiv:1610.09513*.

[Orchard et al. 2015a] Orchard, G.; Meyer, C.; Etienne-Cummings, R.; and Posch, C. 2015a. Hfirst: A temporal approach to object recognition. *IEEE Trans Pattern Anal Mach Intell* 37(10):2028–40.

[Orchard et al. 2015b] Orchard, G.; Jayawant, A.; Cohen, G. K.; and Thakor, N. 2015b. Converting static image

datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience* 9(178).

[Panda and Roy 2016] Panda, P., and Roy, K. 2016. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. In *International Joint Conference on Neural Networks*, 299–306.

[Paszke et al. 2017] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. *NIPS*.

[Rueckauer et al. 2017] Rueckauer, B.; Lungu, I. A.; Hu, Y.; Pfeiffer, M.; and Liu, S. C. 2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification:. *Frontiers in Neuroscience* 11:682.

[Sengupta et al. 2018] Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; and Roy, K. 2018. Going deeper in spiking neural networks: Vgg and residual architectures. *arXiv preprint arXiv:1802.02627*.

[Sironi et al. 2018] Sironi, A.; Brambilla, M.; Bourdis, N.; Lagorce, X.; and Benosman, R. 2018. Hats: Histograms of averaged time surfaces for robust event-based object classification. *arXiv preprint arXiv:1803.07913*.

[Tang, Hua, and Wang 2017] Tang, W.; Hua, G.; and Wang, L. 2017. How to train a compact binary neural network with high accuracy? In *AAAI*, 2625–2631.

[Tavanaei and Maida 2017] Tavanaei, A., and Maida, A. S. 2017. Bio-inspired spiking convolutional neural network using layer-wise sparse coding and stdp learning. *arXiv preprint arXiv:1611.03000*.

[Tavanaei et al. 2018] Tavanaei, A.; Ghodrati, M.; Kheradpisheh, S. R.; Masquelier, T.; and Maida, A. S. 2018. Deep learning in spiking neural networks. *arXiv preprint arXiv:1804.08150*.

[Timothe and Thorpe 2007] Timothe, M., and Thorpe, S. J. 2007. Unsupervised learning of visual features through spike timing dependent plasticity. *Plos Computational Biology* 3(2):e31.

[Wu and He 2018] Wu, Y., and He, K. 2018. Group normalization. *arXiv preprint arXiv:1803.08494*.

[Wu et al. 2018] Wu, Y.; Deng, L.; Li, G.; Zhu, J.; and Shi, L. 2018. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience* 12.

Supplementary material

In this supplementary material, we provide the details of our experiment, including the dataset introduction, training setting, and programing platform comparison.

A   Dataset Introduction

Neuromorphic dataset

N-MNIST converts the frame-based MNIST handwritten digit dataset into its DVS (dynamic vision sensor) version (Orchard et al. 2015b) with event streams (in Fig.8). For each sample, DVS scans the static MNIST image along given directions and collects the generated spike train which is triggered by detecting the intensity change of pixels. Since the intensity change has two directions (increase or decrease), DVS can capture two kinds of spike events, denoted as On-event and Off-event. Because of the relative shift of images during moving process, the pixel dimension is expanded to 34×34. Overall, each sample in N-MNIST is a spatio-temporal spike pattern with size of $34 \times 34 \times 2 \times T$, where T is the length of temporal window.

DVS-CIFAR10 converts 10000 static CIFAR10 images into the format of spike trains (in Fig.8), consisting of 1000 images per class with size of 128×128 for each image . Since different DVS types and different movement paths are used (Li et al. 2017), the generated spike train contains imbalanced spike events and larger image resolution. We adopt different parameter configurations, shown in Tab. 7. DVS-CIFAR10 has 6 times less samples than the original CIFAR10 dataset, and we randomly choose 9000 images for training and 1000 for testing.
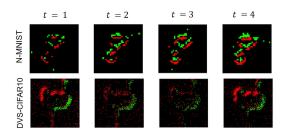

Figure 7: **Illustration of neuromorphic datasets**. The upper 32×32 image and the lower 128×128 image are sampled from N-MNIST and DVS-CIFAR10, respectively. Each sub-picture shows a 5 ms-width spike train.

Non-spiking dataset

CIFAR10 is widely used in ANN domain, which contains 60000 color images belonging to 10 classes with size of 32×32 for each image. We divide it into 50000 training images and 10000 test images as usual.

B   Training setting

Data pre-processing   On N-MNIST, we reduce the time resolution by accumulating the spike train within every 5 ms. On DVS-CIFAR10, we reduce the spatial resolution by down-sampling the original 128×128 size to 42×42 size (stride = 3, padding = 0), and reduce the temporal resolution by similarly accumulating the spike train within every 5 ms. On CIFAR10, as usual, we first crop and flip the original images along each RGB channel, and then rescale each image by subtracting the global mean value of pixel intensity and dividing by the resulting standard variance along each RGB channel.

Optimizer   On neuromorphic datasets, we use Adam (adaptive moment estimation) optimizer. On the non-spiking dataset, we use the stochastic gradient descent (SGD) opti-

mizer with initial learning rate $r = 0.1$ and momentum 0.9, and we let r decay to $0.1r$ over each 40 epochs.

capsulated function modules are unfriendly for users to customize deep SNNs. Therefore, in terms of flexibility, all convolution operations we adopted are based on the officially provided operations (i.e. $conv$ function).

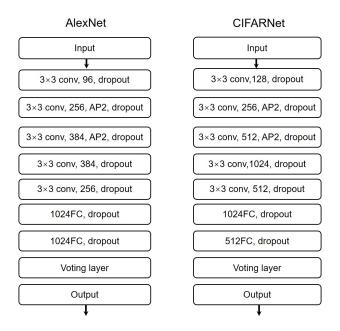| AlexNet	CIFARNet
Input	Input
3×3 conv, 96, dropout	3×3 conv,128, dropout
3×3 conv, 256, AP2, dropout	3×3 conv, 256, AP2, dropout
3×3 conv, 384, AP2, dropout	3×3 conv, 512, AP2, dropout
3×3 conv, 384, dropout	3×3 conv,1024, dropout
3×3 conv, 256, dropout	3×3 conv, 512, dropout
1024FC, dropout	1024FC, dropout
1024FC, dropout	512FC, dropout
Voting layer	Voting layer
Output	Output

Figure 8: **Illustration of network structures**. For simplicity, we denote the left as AlexNet and the right as CIFARNet.

Table 7: Parameter configuration used for model evaluation.

Parameters	CIFAR10	DVS-CIFAR10	N-MNIST
V_{th}	0.75	0.05	0.25
a	1.0	0.1	0.25
$k_{\tau 1}$	0.25	0.35	0.3
$k_{\tau 2}$	0.9	0.9	0.9
Dropout rate	0.5	0	0
Max epoch	150	200	200
Adam	$\lambda, \beta_1, \beta_2 = 0.9, 0.999, 1\text{-}10^{-8}$		

Parameter configuration The configuration of simulation parameters for each dataset is shown in Tab. 7. The structures of what we call AlexNet and CIFARNet are illustrated in Fig. 8.

C Details for Acceleration experiment

A total of 10 experiments are counted for average. Considering the slow running time on the Matlab, 10 simulation steps and batch size of 20 per epoch are used. All codes run on server with i7-6700K CPU and GTX1060 GPU. The Pytorch version is 3.5 and the Matlab version is 2018b. Both of them enable GPU execution, but without parallelization.

It is worth noting that recently there emerge more supports on Matlab for deep learning, either official [2] or unofficial (e.g. MatConvNet toolbox[3]). However, their highly en-

---
[2]Since 2017 version, Matlab provides deep learning libraries.
[3]A MATLAB toolbox designed for implementing CNNs.