SPIDE: A purely spike-based method for training feedback spiking neural networks

Mingqing Xiao [a], Qingyan Meng [b,c], Zongpeng Zhang [d], Yisen Wang [a,e], Zhouchen Lin [a,e,f,*]

[a] National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University, China
[b] The Chinese University of Hong Kong, Shenzhen, China
[c] Shenzhen Research Institute of Big Data, Shenzhen 518115, China
[d] Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University, China
[e] Institute for Artificial Intelligence, Peking University, China
[f] Peng Cheng Laboratory, China

ARTICLE INFO

ABSTRACT

Spiking neural networks (SNNs) with event-based computation are promising brain-inspired models for energy-efficient applications on neuromorphic hardware. However, most supervised SNN training methods, such as conversion from artificial neural networks or direct training with surrogate gradients, require complex computation rather than spike-based operations of spiking neurons during training. In this paper, we study spike-based implicit differentiation on the equilibrium state (SPIDE) that extends the recently proposed training method, implicit differentiation on the equilibrium state (IDE), for supervised learning with purely spike-based computation, which demonstrates the potential for energy-efficient training of SNNs. Specifically, we introduce ternary spiking neuron couples and prove that implicit differentiation can be solved by spikes based on this design, so the whole training procedure, including both forward and backward passes, is made as event-driven spike computation, and weights are updated locally with two-stage average firing rates. Then we propose to modify the reset membrane potential to reduce the approximation error of spikes. With these key components, we can train SNNs with flexible structures in a small number of time steps and with firing sparsity during training, and the theoretical estimation of energy costs demonstrates the potential for high efficiency. Meanwhile, experiments show that even with these constraints, our trained models can still achieve competitive results on MNIST, CIFAR-10, CIFAR-100, and CIFAR10-DVS.

1. Introduction

Spiking neural networks (SNNs) are brain-inspired models that transmit spikes between neurons for event-driven energy-efficient computation. SNNs can be implemented with less energy on neuromorphic hardware (Akopyan et al., 2015; Davies et al., 2018; Pei et al., 2019; Roy, Jaiswal, & Panda, 2019), which is regarded as the third generation of neural network models (Maass, 1997) and is gaining increasing attention as an alternative to artificial neural networks (ANNs).

Different from ANNs, however, directly supervised training of SNNs is a hard problem due to the complex spiking neuron model which is discontinuous. To handle this problem, converting ANNs

to SNNs (Deng & Gu, 2021; Hunsberger & Eliasmith, 2015; Rathi, Srinivasan, Panda, & Roy, 2020; Rueckauer, Lungu, Hu, Pfeiffer, & Liu, 2017; Sengupta, Ye, Wang, Liu, & Roy, 2019; Yan, Zhou, & Wong, 2021), or many direct SNN training methods (Bellec, Salaj, Subramoney, Legenstein, & Maass, 2018; Bohte, Kok, & La Poutre, 2002; Jin, Zhang, & Li, 2018; Kim, Kim and Kim, 2020; Meng, Xiao et al., 2022; Neftci, Mostafa, & Zenke, 2019; Shrestha & Orchard, 2018; Wu, Deng, Li, Zhu, & Shi, 2018; Wu et al., 2019; Xiao, Meng, Zhang, Wang, & Lin, 2021; Zhang & Li, 2019, 2020; Zheng, Wu, Deng, Hu, & Li, 2021) have been proposed to incorporate deep learning into SNNs (Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2019). While these methods can partly solve the problems of unsatisfactory performance or high latency, they require complex computation for gradient calculation or approximation rather than the same spike-based computation as the inference process. They aim at training SNNs with general computational operations and deploying trained models for energy-efficient inference with spiking operations.

As a different direction compared with these methods, it is an important problem to consider if the training process can also

take advantage of the spiking nature of common spiking neurons so that the computation of inference and training is consistent and the training process can also share the energy efficiency with event-driven computation. The consistency of the inference and training computation may also pave a path for directly training SNNs on neuromorphic hardware instead of training models on commonly used computational units and deploying them on chips. A few previous works try to train SNNs with spikes (Guerguiev, Lillicrap, & Richards, 2017; Neftci, Augustine, Paul, & Detorakis, 2017; O'Connor & Welling, 2016; Samadi, Lillicrap, & Tweed, 2017; Thiele, Bichler, & Dupret, 2020; Thiele, Bichler, Dupret, Solinas, & Indiveri, 2019). They either are based on direct feedback alignment (DFA) (Nøkland, 2016) and perform poorly, or require special neuron models (Thiele et al., 2020, 2019) that are hardly supported. Besides, they only focus on feedforward network structures imitated from ANNs, which ignores feedback connections that are ubiquitous in the human brain and enable neural networks to be shallower and more efficient (Kubilius et al., 2019; Xiao et al., 2021). Actually, feedback structures suit SNNs more since SNNs will naturally compute with multiple time steps, which can reuse representations and avoid additional uneconomical costs to unfold along the time as in ANNs (Kim, Li, Park, Venkatesha, & Panda, 2022; Xiao et al., 2021). So training algorithms for feedback SNNs, which may also be degraded for feedforward structures by taking feedback as zero, is worth more exploration.

An ideal SNN training method should tackle the common problems, be suitable for flexible structures (feedforward or feedback), and be with spike-based computation for efficiency and high neuromorphic plausibility. The implicit differentiation on the equilibrium state (IDE) method (Xiao et al., 2021), which is recently proposed to train feedback spiking neural networks (FSNNs), is a promising method that may be generalized to spike-based learning for the requirement. They derive that the forward computation of FSNNs converges to an equilibrium state, which follows a fixed-point equation. Based on it, they propose to train FSNNs by implicit differentiation on this equation, which tackles the common difficulties for SNN training including non-differentiability and large memory costs, and has interesting local update properties. In their method, however, they leverage general root-finding methods rather than spike-based computation to solve implicit differentiation.

In this work, we extend the IDE method to spike-based IDE (SPIDE), which fulfills our requirements and has great potential for energy-efficient training with spike-based computation. We introduce ternary spiking neuron couples and propose to solve implicit differentiation by spikes based on them. Our method is also applicable to feedforward structures by setting the feedback connection as zero. In practice, however, it may require long time steps to stabilize the training with spikes due to the approximation error for gradients. So we further dive into the approximation error from the statistical perspective and propose to simply adjust the reset potential of SNNs to achieve an unbiased estimation of gradients and reduce the estimation variance of SNN computation. With these methods, we can train our models in a small number of time steps, which can further improve the energy efficiency as well as the latency. This demonstrates the strong power of spike-based computation for training SNNs. Our contributions include:

1. We propose the SPIDE method which is the first to train high-performance SNNs by spike-based computation with common neuron models. Specifically, we propose ternary spiking neuron couples and prove that implicit differentiation for gradient calculation can be solved by spikes based on this design. Our method is applicable to both feedback and feedforward structures.

2. We theoretically analyze the approximation error of solving implicit differentiation by spikes, and propose to modify the reset potential to remove the approximation bias and reduce the estimation variance, which enables training in a small number of time steps.

3. Experiments show the low latency and firing sparsity during training, and the theoretical estimation of energy costs demonstrates the great potential for energy-efficient training of SNNs with spike-based computation. The performance on MNIST, CIFAR-10, CIFAR-100 and CIFAR10-DVS are competitive as well.

2. Related work

SNN training methods. Early works seek biologically inspired methods to train SNNs, e.g. spike-time dependent plasticity (STDP) (Diehl & Cook, 2015; Kheradpisheh, Ganjtabesh, Thorpe, & Masquelier, 2018) or reward-modulated STDP (Legenstein, Pecevski, & Maass, 2008). Since the rise of successful ANNs, several works try to convert trained ANNs to SNNs to obtain high performance (Deng & Gu, 2021; Hunsberger & Eliasmith, 2015; Li, Deng, Dong, Gong and Gu, 2021; Rathi et al., 2020; Rueckauer et al., 2017; Sengupta et al., 2019; Stöckl & Maass, 2021; Yan et al., 2021). However, they typically suffer from extremely large time steps (some recent state-of-the-art works largely improved this Bu et al., 2022; Li, Deng et al., 2021; Meng, Yan et al., 2022) and their structures are limited in the scope of ANNs. Others try to directly train SNNs by backpropagation through time (BPTT) and use surrogate derivative for discontinuous spiking functions (Bellec et al., 2018; Deng, Li, Zhang, & Gu, 2022; Fang, Yu, Chen, Huang et al., 2021; Fang, Yu, Chen, Masquelier et al., 2021; Jin et al., 2018; Lee, Delbruck, & Pfeiffer, 2016; Li, Guo et al., 2021; Neftci et al., 2019; Shrestha & Orchard, 2018; Wu et al., 2018, 2019; Zhang & Li, 2019; Zheng et al., 2021) or compute gradient with respect to spiking times (Bohte et al., 2002; Kim, Kim et al., 2020; Zhang & Li, 2020). However, they suffer from approximation errors and large training memory costs, and their optimization with surrogate gradients is not well guaranteed theoretically. Xiao et al. (2021) propose the IDE method to train feedback spiking neural networks, which decouples the forward and backward procedures and avoids the common SNN training problems. Tandem learning (Wu, Chua et al., 2021), ASF (Wu, Zhang et al., 2021), and DSR (Meng, Xiao et al., 2022) also share a similar thought of calculating gradients through spike rates, with the main focus only on feedforward networks with closed-form transformations between successive layers instead of implicit fixed-point equations at equilibrium states. However, all these methods require complex computation during training rather than spike-based operations. A few works focusing on training SNNs with spikes either are based on feedback alignment and limited in simple datasets (Guerguiev et al., 2017; Neftci et al., 2017; O'Connor & Welling, 2016; Samadi et al., 2017), or require special neuron models that require consideration of accumulated spikes for spike generation (Thiele et al., 2020, 2019), which is hardly supported in practice. And they are only applicable to feedforward architectures. Instead, we are the first to leverage spikes with common neuron models to train high-performance SNNs with feedback or feedforward structures. The comparison of different methods is illustrated in Table 1.

Equilibrium of neural networks. The equilibrium of neural networks is first considered by energy-based models, e.g. Hopfield Network (Hopfield, 1982, 1984), which view the dynamics of feedback neural networks as minimizing an energy function toward an equilibrium of the local minimum of the energy. With the energy function, recurrent backpropagation (Almeida, 1987; Pineda, 1987) and more recent equilibrium propagation

Table 1
Comparison of different supervised SNN training methods with respect to performance, latency, structure flexibility, neuron model, spike-based or not, and neuromorphic plausibility.

Method	High perform.	Low latency	Struc. Flexi.	Common neuron model	Spike-based	Neuro. Plaus.
ANN-to-SNN	✓	✓[a]	✗	✓	✗	Low
BPTT with surrogate gradients	✓	✓	✓	✓	✗	Low
DFA with Spikes	✗	?	✗	✓	✓	High
SpikeGrad (Thiele et al., 2020)	✓	?	✗	✗	✓	Medium
IDE (Xiao et al., 2021)	✓	✓	✓	✓	✗	Low
SPIDE (ours)	✓	✓	✓	✓	✓	High

[a]Typical ANN-to-SNN methods require high latency. Some recent state-of-the-art works have largely improved this.

(EP) (Scellier & Bengio, 2017) are proposed to train neural networks. These methods rely on the definition of the energy function and are hardly competitive with deep neural networks. With the steady state of equilibrium, the contrastive Hebbian learning method (Xie & Seung, 2003) is also proposed to train neural networks, which is composed of a two-phase anti-Hebbian and Hebbian procedure and is shown equivalent to backpropagation in the limit of weak feedback. Detorakis, Bartley, and Neftci (2019) propose a variant of this method with random feedback weights which is similar to feedback alignment. However, these methods hardly reach the high performance of backpropagation. Recently, a new branch of deep equilibrium models (Bai, Kolter, & Koltun, 2019; Bai, Koltun, & Kolter, 2020) is proposed. They interpret the computation of weight-tied deep ANNs as solving a fixed-point equilibrium point, and correspondingly propose implicit models which are defined by solving the equilibrium equations and trained by solving implicit differentiation. Their equilibrium is defined by fixed-point equations rather than energy functions and they can achieve superior performance. These works focus on ANNs rather than SNNs, except that Martin et al. (2021), Mesnard, Gerstner, and Brea (2016) and O'Connor, Gavves, and Welling (2019) generalize the idea of EP to SNNs. They follow the energy-based EP method to approximate the gradients, which is different from the equilibrium and the training method in this work, and they are limited to tiny network scales and simple datasets. As for the equilibrium of SNNs, Zhang, Zeng, Zhao and Shi (2018) and Zhang, Zeng, Zhao and Xu (2018) consider the equilibrium state of the membrane potential as an unsupervised learning part, Li and Pehlevan (2020) and Mancoo, Keemink, and Machens (2020) consider the equilibrium from the perspective of solving constrained optimization problems, and Xiao et al. (2021) draw inspiration from deep equilibrium models to view feedback SNNs as evolving along time to an equilibrium state following a fixed-point equation and propose to train FSNNs by implicit differentiation on the equilibrium state (IDE). These methods do not consider training SNNs with spike-based computation. Differently, this work extends IDE to spike-based IDE, which extends the thought of equilibrium of spikes to solving implicit differentiation and enables the whole training procedure to be based on spike computation with common neuron models, providing the potential for energy-efficient training of SNNs.

3. Preliminaries

We first introduce preliminaries about spiking neural network models and the IDE training method (Xiao et al., 2021).

3.1. Spiking neural network models

Spiking neurons draw inspiration from the human brain to communicate with each other by spikes. Each neuron integrates information from input spike trains by maintaining a membrane potential through a differential equation and generates an output spike once the membrane potential exceeds a threshold, following which the membrane potential is reset to the reset potential. We consider the commonly used integrate and fire (IF) and leaky integrate and fire (LIF) neuron models, whose general discretized computational form is:

$$\begin{cases} u_i[t+1] = \lambda(u_i[t] - (V_{th} - u_{reset})s_i[t]) + \sum_j w_{ij}s_j[t] + b_i, \\ s_i[t+1] = H(u_i[t+1] - V_{th}), \end{cases} \quad (1)$$

where $u_i[t]$ is the membrane potential of neuron i at time step t, $s_i[t]$ is the binary output spike train of neuron i, w_{ij} is the connection weight from neuron j to neuron i, b_i is bias, H is the Heaviside step function, V_{th} is the firing threshold, u_{reset} is the reset potential, and $\lambda = 1$ for the IF model while $\lambda < 1$ is a leaky term for the LIF model. We use subtraction as the reset operation. u_{reset} is usually taken as 0 in previous work, while we will reconsider it in Section 4.3. We mainly consider the IF neuron model by default and will demonstrate that our method is applicable to the LIF model as well.

3.2. Background about the IDE training method

Due to the complex spiking neuron model which is discontinuous, directly supervised training of SNNs is a hard problem, since the explicit computation is non-differentiable and therefore backpropagation along the forward computational graph is problematic. The IDE training method (Xiao et al., 2021) considers another approach to calculate gradients that does not rely on the exact reverse of the forward computation, which avoids the problem of non-differentiability as well as large memory costs by BPTT methods with surrogate gradients. Specifically, the IDE training method first derives that the (weighted) average firing rate of FSNN computation with common neuron models would gradually evolve to an equilibrium state along time, which follows a fixed-point equation. Then by viewing the forward computation of FSNN as a black-box solver for this equation, and applying implicit differentiation on the equation, gradients can be calculated only based on this equation and the (weighted) average firing rate during the forward computation rather than the exact forward procedure. Therefore, the forward and backward procedures are decoupled and the non-differentiability is avoided.

We briefly introduce the conclusion of equilibrium states in Section 3.2.1 and the IDE method in Section 3.2.2.

3.2.1. Equilibrium states of FSNNs

Xiao et al. (2021) derive that the (weighted) average rate of spikes during FSNN computation with common neuron models would converge to an equilibrium state following a fixed-point equation given convergent inputs. We first focus on the conclusions with the discrete IF model under both single-layer and multi-layer feedback structures. The single-layer structure has one hidden layer of neurons with feedback connections on this

layer. The update equation of membrane potentials after reset is (let $V_u = V_{th} - u_{reset}$):

$$\mathbf{u}[t+1] = \mathbf{u}[t] + \mathbf{W}\mathbf{s}[t] + \mathbf{F}\mathbf{x}[t+1] + \mathbf{b} - V_u\mathbf{s}[t+1], \quad (2)$$

where $\mathbf{u}[t]$ and $\mathbf{s}[t]$ are the vectors of membrane potentials and spikes of these neurons, $\mathbf{x}[t]$ is the input at time step t, $\mathbf{W}$ is the feedback weight matrix, and $\mathbf{F}$ is the weight matrix from inputs to these neurons. The average input and average firing rate are defined as $\bar{\mathbf{x}}[t] = \frac{1}{t}\sum_{\tau=1}^{t}\mathbf{x}[\tau]$ and $\boldsymbol{\alpha}[t] = \frac{1}{t}\sum_{\tau=1}^{t}\mathbf{s}[\tau]$, respectively. Define $\sigma(x) = \min(1, \max(0, x))$.

The equilibrium state of the single-layer FSNN is described as Xiao et al. (2021): If the average inputs converge to an equilibrium point $\bar{\mathbf{x}}[t] \to \mathbf{x}^*$, and there exists $\gamma < 1$ such that $\|\mathbf{W}\|_2 \leq \gamma V_{th}$, then the average firing rates of FSNN with discrete IF model will converge to an equilibrium point $\boldsymbol{\alpha}[t] \to \boldsymbol{\alpha}^*$, which satisfies the fixed-point equation $\boldsymbol{\alpha}^* = \sigma\left(\frac{1}{V_{th}}(\mathbf{W}\boldsymbol{\alpha}^* + \mathbf{F}\mathbf{x}^* + \mathbf{b})\right)$. Note that they take $u_{reset} = 0$ in this conclusion, if we consider nonzero u_{reset}, the constraint and the fixed-point equation should be $\|\mathbf{W}\|_2 \leq \gamma(V_{th} - u_{reset})$ and $\boldsymbol{\alpha}^* = \sigma\left(\frac{1}{V_{th}-u_{reset}}(\mathbf{W}\boldsymbol{\alpha}^* + \mathbf{F}\mathbf{x}^* + \mathbf{b})\right)$.

The multi-layer structure incorporates more non-linearity into the equilibrium fixed-point equation, which has multiple layers with feedback connections from the last layer to the first layer. The update equations of membrane potentials after reset are expressed as:

$$\begin{cases} \mathbf{u}^1[t+1] = \mathbf{u}^1[t] + \mathbf{W}^1\mathbf{s}^N[t] + \mathbf{F}^1\mathbf{x}[t+1] + \mathbf{b}^1 - V_u\mathbf{s}^1[t+1], \\ \mathbf{u}^l[t+1] = \mathbf{u}^l[t] + \mathbf{F}^l\mathbf{s}^{l-1}[t+1] + \mathbf{b}^l - V_u\mathbf{s}^l[t+1], (l = 2, \ldots, N). \end{cases}$$
$$(3)$$

The equilibrium state of the multi-layer FSNN with u_{reset} is described as Xiao et al. (2021): If the average inputs converge to an equilibrium point $\bar{\mathbf{x}}[t] \to \mathbf{x}^*$, and there exists $\gamma < 1$ such that $\|\mathbf{W}^1\|_2\|\mathbf{F}^N\|_2 \cdots \|\mathbf{F}^2\|_2 \leq \gamma(V_{th} - u_{reset})^N$, then the average firing rates of multi-layer FSNN with discrete IF model will converge to equilibrium points $\boldsymbol{\alpha}^l[t] \to \boldsymbol{\alpha}^{l*}$, which satisfy the fixed-point equations $\boldsymbol{\alpha}^{1*} = f_1(f_N \circ \cdots \circ f_2(\boldsymbol{\alpha}^{1*}), \mathbf{x}^*)$ and $\boldsymbol{\alpha}^{l+1*} = f_{l+1}(\boldsymbol{\alpha}^{l*})$, where $f_1(\boldsymbol{\alpha}, \mathbf{x}) = \sigma\left(\frac{1}{V_{th}-u_{reset}}(\mathbf{W}^1\boldsymbol{\alpha} + \mathbf{F}^1\mathbf{x} + \mathbf{b}^1)\right)$ and $f_l(\boldsymbol{\alpha}) = \sigma\left(\frac{1}{V_{th}-u_{reset}}(\mathbf{F}^l\boldsymbol{\alpha} + \mathbf{b}^l)\right)$.

As for the LIF model, the weighted average firing rate will be considered to represent information: the weighted average input and weighted average firing rate are defined as $\hat{\mathbf{x}}[t] = \frac{\sum_{\tau=1}^{t}\lambda^{t-\tau}\mathbf{x}[\tau]}{\sum_{\tau=1}^{t}\lambda^{t-\tau}}$ and $\hat{\boldsymbol{\alpha}}[t] = \frac{\sum_{\tau=1}^{t}\lambda^{t-\tau}\mathbf{s}[\tau]}{\sum_{\tau=1}^{t}\lambda^{t-\tau}}$, respectively. Xiao et al. (2021) has derived the equilibrium states of FSNN under the LIF model as well. The conclusions are similar to the IF model and the equilibrium states follow the same fixed-point equation, except that weighted average firing rates (inputs) are considered and there would be bounded random error for the equilibrium state. We can view it as an approximate solver for the equilibrium state.

3.2.2. IDE training method

Based on the equilibrium states in Section 3.2.1, we can train FSNNs by calculating gradients with implicit differentiation (Xiao et al., 2021). Let $\boldsymbol{\alpha} = f_\theta(\boldsymbol{\alpha})$ denote the fixed-point equation of the equilibrium state which is parameterized by θ, $g_\theta(\boldsymbol{\alpha}) = f_\theta(\boldsymbol{\alpha}) - \boldsymbol{\alpha}$, and let $\mathcal{L}(\boldsymbol{\alpha}^*)$ denote the objective function with respect to the equilibrium state $\boldsymbol{\alpha}^*$. The implicit differentiation satisfies $\left(\mathbf{I} - \frac{\partial f_\theta(\boldsymbol{\alpha}^*)}{\partial \boldsymbol{\alpha}^*}\right)\frac{d\boldsymbol{\alpha}^*}{d\theta} = \frac{\partial f_\theta(\boldsymbol{\alpha}^*)}{\partial\theta}$ (Bai et al., 2019) (we follow the numerator layout convention for derivatives). Therefore, the differentiation of $\mathcal{L}(\boldsymbol{\alpha}^*)$ for parameters can be calculated based on implicit differentiation as:

$$\frac{\partial\mathcal{L}(\boldsymbol{\alpha}^*)}{\partial\theta} = -\frac{\partial\mathcal{L}(\boldsymbol{\alpha}^*)}{\partial\boldsymbol{\alpha}^*}\left(J_{g_\theta}^{-1}|_{\boldsymbol{\alpha}^*}\right)\frac{\partial f_\theta(\boldsymbol{\alpha}^*)}{\partial\theta}, \quad (4)$$

where $J_{g_\theta}^{-1}|_{\boldsymbol{\alpha}^*}$ is the inverse Jacobian of g_θ evaluated at $\boldsymbol{\alpha}^*$. The calculation of inverse Jacobian can be avoided by solving an alternative linear system (Bai et al., 2019, 2020; Xiao et al., 2021):

$$\left(J_{g_\theta}^{\top}|_{\boldsymbol{\alpha}^*}\right)\boldsymbol{\beta} + \left(\frac{\partial\mathcal{L}(\boldsymbol{\alpha}^*)}{\partial\boldsymbol{\alpha}^*}\right)^{\top} = 0. \quad (5)$$

Note that a readout layer after the last layer of neurons will be constructed for output (Xiao et al., 2021), which is composed of neurons that will not spike and will do classification based on accumulated membrane potentials (e.g. realized by a very large threshold). Then the output o is equivalent to a linear transformation on the approximate equilibrium state, i.e. $\mathbf{o} = \mathbf{W}^o\boldsymbol{\alpha}^* + \mathbf{b}^o$, and the loss will be calculated between $\mathbf{o}$ and labels $\mathbf{y}$ with a common criterion such as cross-entropy. Then the gradient on the equilibrium state can be calculated. For the solution of implicit differentiation, Xiao et al. (2021) follow Bai et al. (2019, 2020) to leverage root-finding methods, while we will solve it by spike-based computation, as will be derived in Section 4. We consider that the (weighted) average firing rates $\boldsymbol{\alpha}[T]$ during the forward computation of FSNNs at time step T roughly reach the equilibrium state. Then by substituting $\boldsymbol{\alpha}^*$ by $\boldsymbol{\alpha}[T]$ in the above equations, gradients for the parameters can be calculated only with $\boldsymbol{\alpha}[T]$ and the equation, and we calculate them based on spikes. With the gradients, first-order optimization methods such as SGD (Rumelhart, Hinton, & Williams, 1986) and its variants can be applied to update parameters.

4. Spike-based implicit differentiation on the equilibrium state

In this section, we present our SPIDE method that calculates the whole training procedure based on spikes. We first introduce ternary spiking neuron couples in Section 4.1 and how to solve implicit differentiation in Section 4.2. Then we theoretically analyze the approximation error and propose the improvement in Section 4.3. Finally, a summary of the training pipeline is presented in Section 4.4. We also provide a brief illustration figure of the overall process in Fig. 1.

4.1. Ternary spiking neuron couples

The common spiking neuron model only generates spikes when the membrane potential exceeds a positive threshold, which limits the firing rate from representing negative information. To enable approximation of possible negative values for implicit differentiation calculation in Section 4.2, we require negative spikes, whose expression could be:

$$s_i[t+1] = T(u_i[t+1], V_{th}) = \begin{cases} 1, & u_i[t+1] > V_{th} \\ 0, & |u_i[t+1]| \leq V_{th} , \\ -1, & u_i[t+1] < -V_{th} \end{cases} \quad (6)$$

and the reset is the same as usual: $u_i[t+1] - (V_{th} - u_{reset})s_i[t+1]$. Direct realization of such ternary output, however, may be not supported by common neuromorphic computation of SNNs.

We propose to leverage two coupled common neurons to realize this computation. As illustrated in Fig. 2, the two coupled neurons with the common IF model (Eq. (1)) receive opposite inputs and output opposite spikes, which aim to deal with positive information and negative information with spikes, respectively. They should share a reset operation in order to accord with Eq. (6), which can be realized by the connections between them: as we use subtraction as the reset operation, the connection whose weight equals $V_{th} - u_{reset}$ enables one neuron to reset the other equivalently. To see how this works, consider the condition
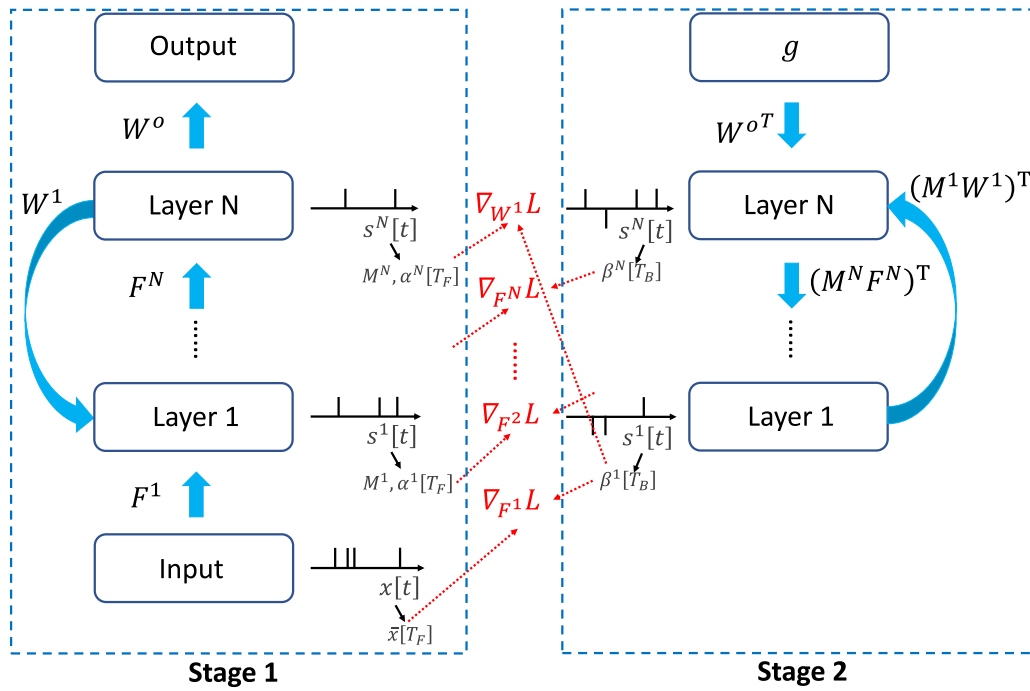
(a) SPIDE
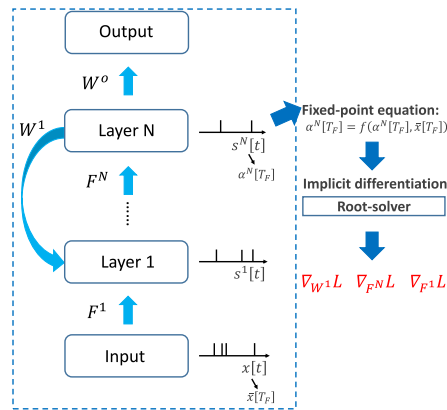
(b) IDE
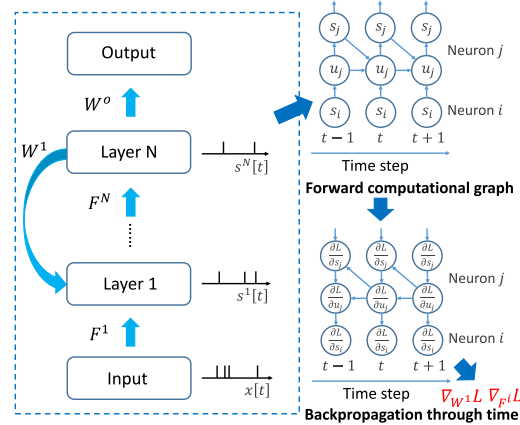
(c) BPTT with SG

Fig. 1. A brief illustration of the overall process of the proposed method and comparison with previous methods. (a) The proposed SPIDE method. Both forward and backward stages are implemented by spike-based computation. (b) The IDE method. The backward stage requires root-solver and complex computation to solve for gradients. (c) The BPTT with surrogate gradient (SG) method. The backward stage requires backpropagation through the unfolded computational graph with complex computation rather than spike operations.
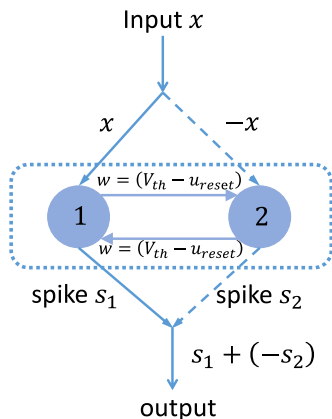

Fig. 2. Illustration of ternary spiking neuron couples.

that the accumulated membrane potential of neuron 1 reaches V_{th}, then neuron 1 would generate a spike and reset, and the output is this positive spike. At the same time, the membrane potential of neuron 2 is $-V_{th}$ and the neuron will not fire and reset, but the spike from neuron 1 will reset it to $-u_{reset}$, which accords with our desired reset for ternary output. Similarly, if the inputs are negative, neuron 2 will generate a spike which will be treated negative as output, and both neurons are reset. For the operation of taking negative, one solution is to enable the reverse operation on hardware; another is to reconnect neuron 2 with other neurons while taking the weight negative to that of neuron 1. As for most existing neuromorphic hardware, the double connections to the two coupled neurons are more feasible, and this implies that twice the synaptic operations are needed for each spike. Therefore, such kind of coupled common neurons can realize ternary output.

We note that the SpikeGrad algorithm (Thiele et al., 2020) also requires neurons for ternary output. However, they do not consider how such kind of operation can be implemented with the common neuron models that follow the basic properties of biological neurons, and moreover, they propose another modified neuron model in practice that requires consideration of accumulated spikes for spike generation, which is hardly supported in practice. Differently, our method is with the common neuron model (IF or LIF). This can be suitable for neuromorphic hardware. Also, SpikingYOLO (Kim, Park, Na and Yoon, 2020) proposes ternary neurons for the inference of SNNs converted from ANNs without considering the implementation by the common neuron models. Our ternary spiking neuron couples may also provide a way to implement their method on hardware.

4.2. Solving implicit differentiation with spikes

Based on the coupled neurons in Section 4.1, we can solve implicit differentiation with spikes. For notation simplicity, we directly use Eq. (6) as a ternary neuron without detailing coupled neurons below. Our main focus is on solving Eq. (5) with spikes. We first consider the IF neuron model by default and will include the LIF model later. The brief outline for the derivation is: we first derive the update equation of membrane potentials, then we derive the equivalent equation of the rate of spikes with eliminating perturbation, and finally, we prove that the firing rate converges to the solution of Eq. (5).

We first consider the single-layer condition. Let $\boldsymbol{\alpha}[T_F]$ denote the average firing rate of these neurons after the forward computation with time steps T_F as an approximate equilibrium state (we treat the forward procedure as the first stage), $\mathbf{g} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}[T_F]}\right)^\top$ denote the gradient of the loss function on this approximate equilibrium state, and $\mathbf{m} = \sigma'(\boldsymbol{\alpha}[T_F])$, $\mathbf{M} = \mathrm{Diag}(\mathbf{m})$ denote a mask indicator based on the firing condition in the first stage, where $\sigma'(x) = \begin{cases} 1, & 0 < x < 1 \\ 0, & \text{else} \end{cases}$. We will have another T_B time steps in the second backward stage to calculate implicit differentiation. We set the input to these neurons as $\mathbf{g}$ at all time steps, which can be viewed as input currents (Xiao et al., 2021; Zhang & Li, 2020). Then along the inverse connections of neurons and with a mask on neurons or weights and an output rescale, the computation of FSNN with ternary neurons is:

$$\mathbf{u}[t+1] = \mathbf{u}[t] + \frac{1}{V_{th} - u_{reset}}(\mathbf{MW})^\top \mathbf{s}[t] + \mathbf{g} - (V_{th}^b - u_{reset}^b)\mathbf{s}[t+1],$$
(7)

where V_{th}, u_{reset} and V_{th}^b, u_{reset}^b are the threshold and reset potential during the first and second stage, respectively. Define the 'average firing rate' at this second stage as $\boldsymbol{\beta}[t] = \frac{1}{t}\sum_{\tau=1}^{t}\mathbf{s}[\tau]$, and $\mathbf{u}[0] = \mathbf{0}, \mathbf{s}[0] = \mathbf{0}$, then through summation, we have (let $V_u = V_{th} - u_{reset}, V_u^b = V_{th}^b - u_{reset}^b$):

$$\boldsymbol{\beta}[t+1] = \frac{1}{V_u^b}\left(\frac{t}{t+1}\frac{1}{V_u}(\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} - \frac{\mathbf{u}[t+1]}{t+1}\right).$$
(8)

Since there would be at most t spikes during t time steps, $\boldsymbol{\beta}$ should be bounded in the range of $[-1, 1]$. The membrane potential $\mathbf{u}_i[t]$ will maintain the exceeded terms, i.e. define $\mathbf{v}_i[t] = \left(\frac{t}{t+1}\frac{1}{V_u}(\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g}\right)_i$, we can divide $\mathbf{u}_i[t]$ as $\mathbf{u}_i^E[t] + \mathbf{u}_i^B[t]$, where $\mathbf{u}_i^E[t] = \max\left(\mathbf{v}_i[t] - V_{th}^b, 0\right) + \min\left(\mathbf{v}_i[t] + V_{th}^b, 0\right)$ is the exceeded term while $\mathbf{u}_i^B[t]$ is a bounded term (Xiao et al., 2021) which is typically bounded in the range of $[-V_{th}^b, V_{th}^b]$. Then, Eq. (8) turns into:

$$\boldsymbol{\beta}[t+1] = \phi\left(\frac{1}{V_u^b}\left(\frac{t}{t+1}\frac{1}{V_u}(\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g}\right)\right) - \frac{1}{V_u^b}\frac{\mathbf{u}^B[t+1]}{t+1},$$
(9)

where $\phi(x) = \min(1, \max(-1, x))$. Note that if $\mathbf{g}$ and $(\mathbf{MW})^\top$ are in an appropriate range, there would be no exceeded term and ϕ will not take effect. Indeed we will rescale the loss to control the range of $\mathbf{g}$, as will be indicated in Section 4.3. With this consideration, we can prove that $\boldsymbol{\beta}[t]$ converges to the solution of Eq. (5).

Theorem 1. *If there exists $\gamma < 1$ such that $\|(\mathbf{MW})^\top\|_2 \leq \gamma(V_{th} - u_{reset})(V_{th}^b - u_{reset}^b)$, then the average firing rate $\boldsymbol{\beta}[t]$ will converge to an equilibrium point $\boldsymbol{\beta}[t] \to \boldsymbol{\beta}^*$. When $V_{th}^b - u_{reset}^b = 1$, and there exists $\lambda < 1$ such that $\|(\mathbf{MW})^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$ and $\|\mathbf{g}\|_\infty \leq 1 - \lambda$, then $\boldsymbol{\beta}^*$ is the solution of Eq. (5).*

The proof and discussion of assumptions are in Appendix A. With Theorem 1, we can solve Eq. (5) by simulating this second stage of SNN computation to obtain the 'firing rate' $\boldsymbol{\beta}[T_B]$ as the approximate solution. Plugging this solution to Eq. (4), the gradients can be calculated by: $\nabla_{\mathbf{W}}\mathcal{L} = \frac{1}{V_{th} - u_{reset}}\mathbf{M}\boldsymbol{\beta}[T_B]\boldsymbol{\alpha}[T_F]^\top$, $\nabla_{\mathbf{F}}\mathcal{L} = \frac{1}{V_{th} - u_{reset}}\mathbf{M}\boldsymbol{\beta}[T_B]\bar{\mathbf{x}}[T_F]^\top$, and $\nabla_{\mathbf{b}}\mathcal{L} = \frac{1}{V_{th} - u_{reset}}\mathbf{M}\boldsymbol{\beta}[T_B]$.

Note that in practice, even if the data distribution is not in a proper range, we can still view ϕ as a kind of clipping for improperly large numbers, which is similar to empirical techniques like "gradient clipping" to stabilize the training.

Then we consider the extension to the multi-layer condition. Let $\boldsymbol{\alpha}^l[T_F]$ denote the average firing rate of neurons in layer l after the forward computation, $\mathbf{g} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}^N[T_F]}\right)^\top$ denote the gradient of the loss function on the approximate equilibrium state of the last layer, and $\mathbf{m}^l = \sigma'(\boldsymbol{\alpha}^l[T_F])$, $\mathbf{M}^l = \mathrm{Diag}(\mathbf{m}^l)$ denote the mask indicators. Similarly, we will have another T_B time steps in the second stage and set the input to the last layer as $\mathbf{g}$ at all time steps. The computation of FSNN with ternary neurons is calculated as:

$$\begin{cases} \mathbf{u}^N[t+1] = \mathbf{u}^N[t] + \frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top \mathbf{s}^1[t] + \mathbf{g} - V_u^b\mathbf{s}^N[t+1], \\ \mathbf{u}^l[t+1] = \mathbf{u}^l[t] + \frac{1}{V_u}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top \mathbf{s}^{l+1}[t+1] \\ \qquad - V_u^b\mathbf{s}^l[t+1], (l = N-1, \ldots, 1). \end{cases}$$
(10)

The 'average firing rates' $\boldsymbol{\beta}^l[t]$ are similarly defined for each layer, and the equivalent form can be similarly derived as:

$$\begin{cases} \boldsymbol{\beta}^N[t+1] = \phi\left(\frac{1}{V_u^b}\left(\frac{t}{t+1}\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top \boldsymbol{\beta}^1[t] + \mathbf{g}\right)\right) - \frac{1}{V_u^b}\frac{\mathbf{u}^{N^B}[t+1]}{t+1}, \\ \boldsymbol{\beta}^l[t+1] = \phi\left(\frac{1}{V_u^b}\left(\frac{1}{V_u}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top \boldsymbol{\beta}^{l+1}[t+1]\right)\right) - \frac{1}{V_u^b}\frac{\mathbf{u}^{l^B}[t+1]}{t+1}. \end{cases}$$
(11)

The convergence of the 'firing rate' at the last layer to the solution of Eq. (5) can be similarly derived as Theorem 1. However, we need to calculate gradients for each parameter as Eq. (4), which is more complex than the single-layer condition. Actually, we can derive that the 'firing rates' at each layer converge to equilibrium points, based on which the gradients can be easily calculated with information from the adjacent layers. Theorem 2 gives a formal description.

Theorem 2. *If there exists $\gamma < 1$ such that $\|(\mathbf{M}^1\mathbf{W}^1)^\top\|_2\| (\mathbf{M}^N \mathbf{F}^N)^\top \|_2 \cdots \|(\mathbf{M}^2\mathbf{F}^2)^\top\|_2 \leq \gamma(V_{th} - u_{reset})^N(V_{th}^b - u_{reset}^b)^N$, then the average firing rates $\boldsymbol{\beta}^l[t]$ will converge to equilibrium points $\boldsymbol{\beta}^l[t] \to \boldsymbol{\beta}^{l*}$. When $V_{th}^b - u_{reset}^b = 1$, and there exists $\lambda < 1$ such that $\|(\mathbf{M}^1\mathbf{W}^1)^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$, $\|(\mathbf{M}^l\mathbf{F}^l)^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$, $l = 2, \ldots, N$ and $\|\mathbf{g}\|_\infty \leq 1 - \lambda^N$, then $\boldsymbol{\beta}^{N*}$ is the solution of Eq. (5),*

and $\boldsymbol{\beta}^{l*} = \left(\frac{\partial h_N(\boldsymbol{\alpha}^{N*})}{\partial h_l(\boldsymbol{\alpha}^{N*})}\right)^\top \boldsymbol{\beta}^{N*}$, $l = N - 1, \ldots, 1$, where $h_l(\boldsymbol{\alpha}^{N*}) = f_l \circ \cdots \circ f_2 \left(f_1(\boldsymbol{\alpha}^{N*}, \mathbf{x}^*)\right)$, $l = N, \ldots, 1$.

The functions f_l are defined in Section 3.2.1. For the proof please refer to Appendix B. With Theorem 2, by putting the solutions into Eq. (4), the gradients can be calculated by: $\nabla_{\mathbf{F}^l}\mathcal{L} = \frac{1}{V_{th}-u_{reset}}\mathbf{M}^l\boldsymbol{\beta}^l[T_B]\boldsymbol{\alpha}^{l-1}[T_F]^\top$ $(l = 2, \ldots, N)$, $\nabla_{\mathbf{F}^1}\mathcal{L} = \frac{1}{V_{th}-u_{reset}}\mathbf{M}^1\boldsymbol{\beta}^1$ $[T_B]\bar{\mathbf{x}}[T_F]^\top$, $\nabla_{\mathbf{W}^1}\mathcal{L} = \frac{1}{V_{th}-u_{reset}}\mathbf{M}^l\boldsymbol{\beta}^l[T_B]\boldsymbol{\alpha}^N[T_F]^\top$, and $\nabla_{\mathbf{b}^l}\mathcal{L} = \frac{1}{V_{th}-u_{reset}}$ $\mathbf{M}^l\boldsymbol{\beta}^l[T_B]$ $(l = 2, \ldots, N)$.

Note that the gradient calculation shares an interesting local property, i.e. it is proportional to the firing rates of the two neurons connected by it: $\nabla_{\mathbf{F}^l_{i,j}}\mathcal{L} = \frac{1}{V_{th}-u_{reset}}\mathbf{m}^l_i\boldsymbol{\beta}^l_i\boldsymbol{\alpha}^{l-1}_j$. During calculation, since we will have the firing rate of the first stage before the second stage, this calculation can also be carried out by event-based calculation triggered by the spikes in the second stage. So the weight update could be event-driven as well.

Also, note that the theorems still hold if we degrade our feedback models to feedforward ones by setting feedback connections as zero. In this setting, the dynamics and equilibriums degrade to direct functional mappings, and the implicit differentiation degrades to the explicit gradient. We can still approximate gradients with this computation.

In the following, we take $V^b_{th} - u^b_{reset} = 1$ by default to fulfill the assumption of theorems (it may take other values if we correspondingly rescale the outputs and we set 1 for simplicity). Other techniques like dropout can also be included. Please refer to Appendix C for details.

SPIDE with LIF neuron model. The above conclusions show that we can leverage the equilibrium states with the IF neuron model to solve implicit differentiation with spikes. As introduced in Section 3.2.1, we can also derive the equilibrium states with the LIF neuron model by considering the weighted average firing rate, and the equilibrium fixed-point equations would have the same form as those of the IF model except for some bounded random error. Therefore, with the same thought, our SPIDE method can leverage LIF neurons to approximate the solution of implicit differentiation as well. We can replace IF neurons and average firing rates with LIF neurons and weighted average firing rates, and it will gradually approximate the same equilibrium states as in Theorems 1 and 2 with bounded random error. The derivation is similar to Xiao et al. (2021) and Theorems, and the repetitive details are omitted here.

4.3. Reducing approximation error

Section 4.2 derives that we can solve implicit differentiation with spikes, as the average firing rate will gradually converge to the solution. In practice, however, we will simulate SNNs for finite time steps, and a smaller number of time steps is better for lower energy consumption. This will introduce approximation error which may hamper training. In this subsection, we theoretically study the approximation error and propose to adjust the reset potential to reduce it. Inspired by the theoretical analysis on quantized gradients (Chen, Gai, Yao, Mahoney, & Gonzalez, 2020), we will analyze the error from the statistical perspective.

For the 'average firing rates' $\boldsymbol{\beta}^l[t]$ in Eq. (8) and the multilayer counterparts, the approximation error e to the equilibrium states consists of three independent parts e_e, e_r and e_i: the first is $\mathbf{u}^{l^E}[t + 1]$ that is the exceeded term due to the limitation of spike number, the second is $\mathbf{u}^{l^B}[t + 1]$ which can be viewed as a bounded random variable, and the third is the convergence error of the iterative update scheme without $\mathbf{u}^l[t + 1]$, i.e. let $\mathbf{b}^l[t]$ denote the iterative sequences for solving $\boldsymbol{\beta}^{l*}$ as $\mathbf{b}^l[t + 1] = \frac{t}{t+1}\frac{1}{V_{th}-u_{reset}}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top\mathbf{b}^l[t]$, the convergence error is $\|\mathbf{b}^l[t] -$

$\boldsymbol{\beta}^{l*}\|$. The second part e_r can be again decomposed into two independent components $e_r = e_q + e_s$: e_q is the quantization effect due to the precision of firing rates ($\frac{1}{T}$ for T time steps) if we first assume the same average inputs at all time steps, and e_s is due to the random arrival of spikes rather than the average condition, as there might be unexpected output spikes, e.g. the average input is 0 and the expected output should be 0, but two large positive inputs followed by one larger negative input at the last time would generate two positive spikes while only one negative spike. So the error is divided into: $e = e_e + e_q + e_s + e_i$. Since the iterative formulation is certain for e_i, we focus on e_e, e_q and e_s.

Firstly, the error e_q due to the quantization effect is influenced by the input scale and time steps T_B. To enable proper input scale and smaller time steps, we rescale the loss function by a factor s_l, since the magnitude of gradients with cross-entropy loss is relatively small. We scale the loss to an appropriate range so that information can be propagated by SNNs in smaller time steps, and most signals are in the range of ϕ as analyzed in Section 4.2. The base learning rate is scaled by $\frac{1}{s_l}$ correspondingly. This is also adopted by Thiele et al. (2020).

Then given the scale and number of time steps, e_q, e_e and e_s can be treated as random variables from statistical perspective, and we view $\boldsymbol{\beta}^l[t]$ as stochastic estimators for the equilibrium states with e_i. For the stochastic optimization algorithms, the expectation and variance of the gradients are important for convergence and convergence rate (Bottou, 2010), i.e. we hope an unbiased estimation of gradients and smaller estimation variance. As for e_e and e_s, they depends on the input data and the expectations are $\mathbb{E}[e_e] = 0$, $\mathbb{E}[e_s] = 0$ (the positive and negative parts have the same probability). While for e_q, it will depend on our hyperparameters V^b_{th} and u^b_{reset}. Since the remaining terms in $\mathbf{u}^{l^B}[t + 1]$ caused by the quantization effect is in the range of $[u^b_{reset}, V^b_{th}]$ for positive terms while $[-V^b_{th}, -u^b_{reset}]$ for negative ones, given $V^b_{th} - u^b_{reset}$ and considering the uniform distribution, only when $u^b_{reset} = -V^b_{th}$, $\mathbb{E}[e_q] = 0$ for both positive and negative terms. Therefore, we should adjust the reset potential from commonly used 0 to $-V^b_{th}$ for unbiased estimation, as described in Proposition 3.

Proposition 3. *For fixed $V^b_{th} - u^b_{reset}$ and uniformly distributed inputs and e_q, only when $u^b_{reset} = -V^b_{th}$, $\boldsymbol{\beta}^l[t]$ are unbiased estimators for $\mathbf{b}^l[t]$.*

Also, taking $u^b_{reset} = -V^b_{th}$ achieves the smallest estimation variance for the quantization effect e_q, considering the uniform distribution on $[u^b_{reset}, V^b_{th}] \cup [-V^b_{th}, -u^b_{reset}]$. Since the effects of e_e, e_s and e_i are independent of e_q and their variance is certain given inputs, it leads to Proposition 4.

Proposition 4. *Taking $u^b_{reset} = -V^b_{th}$ reduces the variance of estimators $\boldsymbol{\beta}^l[t]$.*

With this analysis, we will take $V^b_{th} = 0.5$, $u^b_{reset} = -0.5$ in the following. For V_{th} and u_{reset} during the first forward stage, we will also take $u_{reset} = -V_{th}$.

4.4. Details and training pipeline

The original IDE method (Xiao et al., 2021) leverages other training techniques including modified batch normalization (BN) and restriction on weight spectral norm. Since the batch statistical information might be hard to obtain for calculation of biological systems and neuromorphic hardware, and training with the BN operation is hard to be implemented by spike-based calculation, we drop BN in our SPIDE method. The restriction on

Table 2
Evaluation of training with different time steps in the backward stage. Training is on CIFAR-10 with AlexNet-F structure and $T_F = 30$. Results are based on 3 runs of experiments.

T_B	Mean ± Std (Best)
50	88.41% ± 0.48% (89.07%)
100	89.17% ± 0.14% (89.35%)
250	89.61% ± 0.11% (89.70%)
500	89.57% ± 0.08% (89.67%)

the weight norm, however, is necessary for the convergence of feedback models, as indicated in theorems. We will adjust it for a more friendly calculation, please refer to Appendix C for details.

Algorithm 1 Forward procedure of SPIDE training - Stage 1.

Input: Network parameters $F^1, b^1, \cdots, F^N, b^N, W^1, W^o, b^o$; Time steps T_F; Forward threshold V_{th}; Dropout rate r; Input data $\{x[t]\}_{t=1}^{T_F}$;
Output: Output of the readout layer o.

1: Initialize $u^i[0] = 0 (i = 1, 2, \cdots, N)$, $o = 0$
2: If use dropout, randomly generate dropout masks $D^i(i = 1, 2, \cdots, N), D^f$ with rate r // D^i, D^f are saved for backward
3: **for** $t = 1, 2, \cdots, T_F$ **do**
4: **if** $t == 1$ **then**
5:    $u^1[t] = u^1[t-1] + D^1 \odot (F^1 x[t] + b^1)$
6: **else**
7:    $u^1[t] = u^1[t-1] + D^1 \odot (F^1 x[t] + b^1) + D^f \odot (W^1 s^N[t-1])$
8: $s^1[t] = H(u^1[t] - V_{th})$
9: $u^1[t] = u^1[t] - 2V_{th}s^1[t]$ // $u_{reset} = -V_{th}$, the same below
10: **for** $l = 2, 3, \cdots, N$ **do**
11:    $u^l[t] = u^l[t-1] + D^l \odot (F^l s^{l-1}[t] + b^l)$
12:    $s^l[t] = H(u^l[t] - V_{th})$
13:    $u^l[t] = u^l[t] - 2V_{th}s^l[t]$
14: $o = o + W^o s^N[t] + b^o$ // o can be accumulated here, or calculated later by $o = W^o \alpha^N + b^o$
15: $o = \frac{o}{T}$
16: $\alpha^i = \frac{\sum_{t=1}^{T} s^i[t]}{T}, i = 1, 2, \cdots, N$ // Save for backward, firing rate in Stage 1
17: $m^i = (\alpha^i > 0) \wedge (\alpha^i < 1)$ // Save for backward, mask
18: If x is not constant, save $x = \frac{\sum_{t=1}^{T} x[t]}{T}$ for backward
19: **Return** o

We summarize our training pipeline as follows. There are two stages for the forward and backward procedures respectively. In the first stage, SNNs receive inputs and perform the calculation as Eqs. (1), (2), (3) for T_F time steps, after which we get the output from the readout layer, and save the average inputs as well as the average firing rates and masks of each layer for the second stage. In the second stage, the last layer of SNNs will receive gradients for outputs and perform calculation along the inverse connections as Eqs. (6), (7), (10) for T_B time steps, after which we get the 'average firing rates' of each layer. With firing rates from two stages, the gradients for parameters can be calculated as in Section 4.2 and then the first-order optimization algorithm is applied to update the parameters. We provide detailed pseudocodes for both stages in Algorithms 1 and 2, respectively. Fig. 1 also illustrates the overall process.

5. Experiments

In this section, we conduct experiments to demonstrate the effectiveness of our method and the great potential for energy-efficient training. We simulate the computation on common computational units. Please refer to Appendix C for implementation details and descriptions.

Effectiveness with a small number of backward time steps. As shown in Table 2, we can train high-performance models with low latency ($T_F = 30$) in a small number of backward time steps during training (e.g. $T_B = 50$), which indicates the low latency and

---

Algorithm 2 Backward procedure of SPIDE training - Stage 2.

Input: Network parameters $F^1, b^1, \cdots, F^N, b^N, W^1, W^o, b^o$; Forward output o; Label y; Time steps T_B; Forward threshold V_{th}; Backward threshold $V_{th}^b = 0.5$; Other hyperparameters and saved variables;
Output: Trained network parameters $F^1, b^1, \cdots, F^N, b^N, W^1, W^o, b^o$.

1: Calculate $g = \frac{\partial L(o,y)}{\partial o}$ // for CE loss, $\frac{\partial L(o,y)}{\partial o} = \text{softmax}(o) - y$, in practice we will scale the loss by a factor s_l, then $\frac{\partial L(o,y)}{\partial o} = s_l(\text{softmax}(o) - y)$
2: Initialize $u^i[0] = 0, i = 1, 2, \cdots, N$
3: **for** $t = 1, 2, \cdots, T_B$ **do**
4: **if** $t == 1$ **then**
5:    $u^N[t] = u^N[t-1] + W^{o\top} g$
6: **else**
7:    $u^N[t] = u^N[t-1] + W^{o\top} g + \frac{1}{2V_{th}} W^{1\top} (D^f \odot m^1 \odot s^1[t-1])$ // m^i is the saved mask in Stage 1
8: $s^N[t] = T(u^N[t], 0.5)$ // realized by two coupled neurons
9: $u^N[t] = u^N[t] - s^N[t]$ // realized by two coupled neurons
10: **for** $l = N - 1, N - 2, \cdots, 1$ **do**
11:    $u^l[t] = u^l[t-1] + \frac{1}{2V_{th}} F^{l\top} (D^l \odot m^{l+1} \odot s^{l+1}[t])$ // m^i is the saved mask in Stage 1
12:    $s^l[t] = T(u^l[t], 0.5)$ // realized by two coupled neurons
13:    $u^l[t] = u^l[t] - s^l[t]$ // realized by two coupled neurons
14: $\beta^i = \frac{\sum_{t=1}^{T} s^i[t]}{T}, i = 1, 2, \cdots, N$ // "firing rate" in Stage 2
15: Calculate gradients: // Note that the below calculation can be realized by event-driven accumulation based on the above $s^i[t]$ considering the definition of β^i
16: (1) $\nabla_{F^1} \mathcal{L} = \frac{1}{2V_{th}} (m^1 \odot \beta^1) x^\top$ // m^i, x are the saved mask and average input in Stage 1
17: (2) $\nabla_{F^i} \mathcal{L} = \frac{1}{2V_{th}} (m^i \odot \beta^i) \alpha^{i-1\top}, i = 2, 3, \cdots, N$ // m^i, α^i are the saved mask and firing rate in Stage 1
18: (3) $\nabla_{b^i} \mathcal{L} = \frac{1}{2V_{th}} (m^i \odot \beta^i), i = 1, 2, \cdots, N$
19: (4) $\nabla_{W^1} \mathcal{L} = \frac{1}{2V_{th}} (m^1 \odot \beta^1) \alpha^{N\top}$ // m^i, α^i are the saved mask and firing rate in Stage 1
20: (5) $\nabla_{W^o} \mathcal{L} = \alpha^N \left(\frac{\partial L(o,y)}{\partial o} \right)^\top$ // α^i is the saved firing rate in Stage 1
21: (6) $\nabla_{b^o} \mathcal{L} = \left(\frac{\partial L(o,y)}{\partial o} \right)^\top$
22: Update $F^1, b^1, \cdots, F^N, b^N, W^1, W^o, b^o$ based on the gradient-based optimizer // SGD learning rate η + momentum α & weight decay μ, the base learning rate is scaled by the factor s_l of the loss, i.e. $\eta = \frac{\eta}{s_l}$
23: (1) Update the momentum $M_\theta = \alpha * M_\theta + (1 - \alpha) * \nabla_\theta \mathcal{L}, \theta \in \{F^i, b^i, W^1, W^o, b^o\}$
24: (2) Update parameters $\theta = (1 - \mu) * \theta + \eta * M_\theta, \theta \in \{F^i, b^i, W^1, W^o, b^o\}$
25: (3) Restrict the norm of W^1
26: **Return** $F^1, b^1, \cdots, F^N, b^N, W^1, W^o, b^o$

---

high energy efficiency. Note that conventional ANN-SNN methods require hundreds to thousands of time steps just for satisfactory inference performance, and recent progress of the conversion methods and direct training methods show that relatively small time steps are enough for inference, while we are the first to demonstrate that even training of SNNs can be carried out with spikes in a very small number of time steps. This is due to our analysis and improvement to reduce the approximation error, as illustrated in the following ablation study.

Ablation study of reducing the approximation error. We conduct ablation study on our improvement to reduce the approximation error by setting the reset potential as negative threshold. To formulate equivalent equilibrium states, we take the same $V_{th} - u_{reset} = V_u$ and the same $V_{th}^b - u_{reset}^b = V_u^b$, and we consider the following settings: (1) both forward and backward stages apply our improvement, i.e. $u_{reset} = -V_{th}, u_{reset}^b = -V_{th}^b$; (2) remove the improvement on the backward stage, i.e. $V_{th}^b = V_u^b, u_{reset}^b = 0$; (3) remove the improvement on both forward and backward stages, i.e. $V_{th} = V_u, u_{reset} = 0$ and $V_{th}^b = V_u^b, u_{reset}^b = 0$. The latter two setting are denoted by "w/o B" and "w/o F&B" respectively. The models are trained on CIFAR-10 with AlexNet-F structure and 30 forward time steps. The training and testing curves under different settings and backward time steps are illustrated in Fig. 3. It demonstrates that without our improvement, the
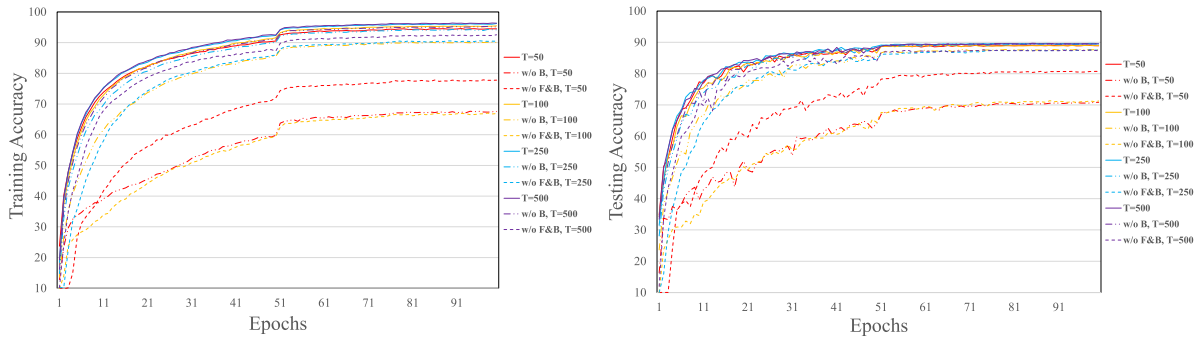
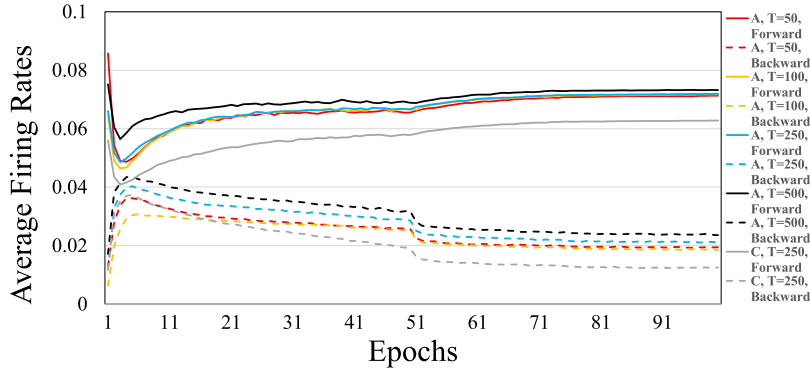**Fig. 3.** Comparison of training and testing curves under different settings and backward time steps.

Fig. 4. Average firing rates for forward and backward stages during training. 'A' is AlexNet-F, 'C' is CIFARNet-F, and T is time steps for the backward stage.

Table 3
Theoretical estimation of energy costs of the backward stage of training with $T_F = 30, T_B = 50$.

Method	OpNum	OpEnergy	Cost
SPIDE (ours)	$\approx 0.03 \times 50$ (FiringRate $\times T_B$)	0.9 (AC)	1×
STBP	30 (T_F)	4.6 (MAC)	102×

Table 4
Theoretical estimation of energy costs of the forward stage with $T_F = 30$.

Method	OpNum	OpEnergy	Cost
Spike-based	$\approx 0.07 \times 30$ (FiringRate $\times T_F$)	0.9 (AC)	1×
Not spike-based	30 (T_F)	4.6 (MAC)	73×

training cannot perform well within a small number of backward time steps, probably due to the bias and large variance of the estimated gradients. When the backward time steps are large, the performance gap is reduced since the bias of estimation is reduced. It shows the superiority of our improvement in training SNNs within a small number of backward time steps.

Firing rate statistics and potential of energy efficiency. Since the energy consumption of event-driven SNNs is proportional to the number of spikes, we present the average firing rates for forward and backward stages (for backward, both positive and negative spikes are considered as firing) in Fig. 4. It shows the firing sparsity of our method, and spikes are sparser in the backward stage with around only 3%. Combined with the small number of time steps, this demonstrates the great potential for the energy-efficient training of SNNs with spike-based computation. We theoretically estimate and compare the energy costs for the operations of neurons of our method and the representative STBP

method.[1] We mainly focus on the energy of the backward stage which is made spike-based and only requires accumulation (AC) operations by SPIDE while requiring multiply and accumulate (MAC) operations by STBP. Our operation number is estimated as the firing rate multiplied by backward time steps T_B, and that of STBP is the forward time steps T_F as STBP will repeat the computation for T_F times to backpropagate through time. According to the 45 nm CMOS processor, the energy for 32 bit FP MAC operation is 4.6 pJ, and for AC operation is 0.9 pJ. Therefore, as shown in Table 3, when $T_F = 30, T_B = 50$, our method could achieve approximately 102× reduction in energy cost.[2] Additionally, if we consider that the forward computation of STBP also has to be simulated by, e.g. GPU, to be compatible with the backward stage and the computation is not spike-based, while ours may be deployed with neuromorphic hardware for spike-based computation, then the forward stage during training can also reduce the energy by around 73×, as shown in Table 4. Note that even for STBP with fewer forward time steps, e.g. 12 or 6, the energy costs of SPIDE for the backward stage are still about 40× or 20× less than STBP, and our result in Table 6 will show that SPIDE is also effective with smaller T_F.

---

[1] The energy costs of the IDE method are hard to estimate because IDE leverages root-finding methods (e.g. Broyden's method) to solve implicit differentiation, which may involve many complex operations and the iteration number may depend on the convergence speed. If we consider the fixed-point update scheme with a fixed iteration number N as the root-finding method in the IDE, its operation number is about the same as STBP with N time steps to backpropagate. As the iteration threshold of IDE is usually taken as 30 (i.e. the maximum iteration number for root-finding methods if the update does not converge before it), the energy estimation of STBP with 30 time steps can also be an effective surrogate result for IDE.

[2] Note that the realization of ternary spiking neuron couples may require twice the synaptic operations under some cases (see Section 4.1), so the energy cost reduction may be halved. Despite this, we could still achieve approximately 50× reduction in energy cost.

Table 5
Performance on MNIST with 3 runs of experiments. "N.A." means the method is not spike-based.

MNIST						
Method	Network structure	T_F	T_B	Mean $\pm$ Std (Best)	Neurons	Params
BP (Lee et al., 2016)	20C5-P2-50C5-P2-200	>200	N.A.	(99.31%)	33k	518k
STBP (Wu et al., 2018)	15C5-P2-40C5-P2-300	30	N.A.	(99.42%)	26k	607k
IDE (Xiao et al., 2021)	64C5 (F64C5)	30	N.A.	99.53% $\pm$ 0.04% (99.59%)	13k	229k
SpikeGrad (Thiele et al., 2020)	15C5-P2-40C5-P2-300	Unknown	Unknown	99.38% $\pm$ 0.06% (99.52%)	26k	607k
SPIDE (ours)	64C5s-64C5s-64C5 (F64C3u)	30	100	99.34% $\pm$ 0.02% (99.37%)	20k	275k
SPIDE (ours, degraded)	15C5-P2-40C5-P2-300	30	100	99.44% $\pm$ 0.02% (99.47%)	26k	607k

Table 6
Performance on CIFAR-10 with 3 runs of experiments. "N.A." means the method is not spike-based.

CIFAR-10						
Method	Network structure	T_F	T_B	Mean $\pm$ Std (Best)	Neurons	Params
ANN-SNN (Sengupta et al., 2019)	VGG-16	2500	N.A.	(91.55%)	311k	15M
ANN-SNN (Deng & Gu, 2021)	CIFARNet	400–600	N.A.	(90.61%)	726k	45M
STBP (Wu et al., 2019)	AlexNet	12	N.A.	(85.24%)	595k	21M
STBP (w/o NeuNorm) (Wu et al., 2019)	CIFARNet	12	N.A.	(89.83%)	726k	45M
STBP (Xiao et al., 2021)	AlexNet-F	30	N.A.	(87.18%)	159k	3.7M
IDE (Xiao et al., 2021)	AlexNet-F	30	N.A.	91.74% $\pm$ 0.09% (91.92%)	159k	3.7M
IDE (Xiao et al., 2021)	CIFARNet-F	30	N.A.	92.08% $\pm$ 0.14% (92.23%)	232k	11.8M
SpikeGrad (Thiele et al., 2020)	CIFARNet	Unknown	Unknown	89.49% $\pm$ 0.28% (89.99%)	726k	45M
SPIDE (ours)	AlexNet-F	12	250	89.11% $\pm$ 0.29% (89.43%)	159k	3.7M
SPIDE (ours)	AlexNet-F	30	250	89.61% $\pm$ 0.11% (89.70%)	159k	3.7M
SPIDE (ours)	CIFARNet-F	30	250	89.94% $\pm$ 0.17% (90.13%)	232k	11.8M

Competitive performance on common datasets. We evaluate the performance of our method on static datasets MNIST (LeCun, Bottou, Bengio, & Haffner, 1998), CIFAR-10, and CIFAR-100 (Krizhevsky & Hinton, 2009), as well as the neuromorphic dataset CIFAR10-DVS (Li, Liu, Ji, Li, & Shi, 2017). We compare our method to several ANN-SNN methods (Deng & Gu, 2021; Sengupta et al., 2019), direct SNN training methods (Wu et al., 2018; Xiao et al., 2021), and SpikeGrad (Thiele et al., 2020) with similar network structures. As shown in Tables 5 and 6, we can train both feed-forward and feedback SNN models with a small number of time steps and our trained models achieve competitive results on MNIST and CIFAR-10. Compared with SpikeGrad (Thiele et al., 2020), we can use fewer neurons and parameters due to flexible network structure choices, and a small number of time steps while they do not report this important feature. Besides, we use common neuron models while they require special neuron models that are hardly supported, as indicated in Section 4.1. Compared with the original IDE method (Xiao et al., 2021), our generalization performance is poorer as we discard the BN component. As shown in Fig. 5, the SPIDE method can achieve the same training accuracy as the IDE method, while the generalization performance is poorer. Since the hyperparameters are the same for experiments, except that we drop the modified BN component (as explained in Section 4.4), the performance gap may be caused by the implicit regularization effect of BN. We try to drop the modified BN component for IDE and the results in Fig. 5 show that it achieves similar performance (89.93%) as SPIDE. Therefore, the optimization ability of the SPIDE method should be similar to that of IDE, while future work could investigate techniques that are similar to BN but more friendly to the spike-based computation of SNNs to further improve the performance.

The results on CIFAR-100 are shown in Table 7 and our model can achieve 64.07% accuracy. Compared with IDE, the performance is poorer, and the main reason is probably again the absence of BN which could be important for alleviating overfitting on CIFAR-100 with a relatively small number of images per class. The training accuracy of SPIDE is similar to IDE (around 93% v.s. around 94%) while the generalization performance is poorer. Despite this, the performance of our model is competitive

for networks without BN and our model is with fewer neurons and parameters and a small number of time steps. Compared with SpikeGrad (Thiele et al., 2020), we can use fewer neurons and parameters due to flexible network structure choices, and we leverage common neuron models while they do not. Future work could investigate more suitable structures and more friendly techniques to further improve the performance.

The results on CIFAR10-DVS are shown in Table 8, and our model can achieve 60.7% accuracy. It is competitive among results of common SNN models, demonstrating the effectiveness of our method.

The above results show the effectiveness of our method even with the constraint of purely spike-based training. We note that there are some recent works that achieve higher state-of-the-art performance (Deng et al., 2022; Fang, Yu, Chen, Huang et al., 2021; Fang, Yu, Chen, Masquelier et al., 2021; Li, Guo et al., 2021; Zheng et al., 2021). However, their target is different from ours which aims at training SNNs with purely spike-based computation as introduced in Section 1, and they leverage many other techniques such as batch normalization along the temporal dimension or learnable membrane time constant. We do not aim at outperforming the state-of-the-art results but demonstrate that a competitive performance can be achieved even with our constraints of purely spike-based training with common neuron models. And our future work could seek techniques friendly to neuromorphic computation to further improve the performance.

As a preliminary attempt, we provide the result of applying the scaled weight standardization (sWS) technique which is shown as a powerful method to replace BN in ResNets (Brock, De and Smith, 2021; Brock, De, Smith and Simonyan, 2021) and SNNs (Xiao, Meng, Zhang, He, & Lin, 2022). Particularly, sWS standardizes weights instead of activations by $\hat{\mathbf{W}}_{i,j} = \gamma \cdot \frac{\mathbf{W}_{i,j} - \mu_{\mathbf{W}_{i,\cdot}}}{\sigma_{\mathbf{W}_{i,\cdot}} \cdot \sqrt{N}}$, where $\mu_{\mathbf{W}_{i,\cdot}}$ and $\sigma_{\mathbf{W}_{i,\cdot}}$ are the mean and variance calculated along the input dimension, and the scale γ is determined by analyzing the signal propagation with different activation functions (typically taken as $\gamma = \frac{\sqrt{2}}{\sqrt{1 - \frac{1}{\pi}}}$). Brock, De, Smith (2021) show that normalization-free ResNets with the sWS technique can achieve a similar performance as common ResNets with BN. We apply
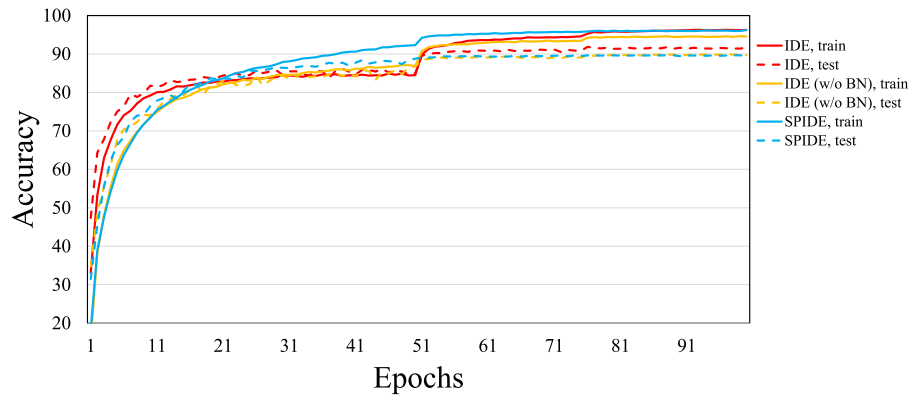
Fig. 5. Comparison of training and testing curves between IDE and SPIDE on CIFAR-10 with AlexNet-F structure and $T_F = 30$.

Table 7
Performance on CIFAR-100 with 3 runs of experiments. "N.A." means the method is not spike-based.

Method	Network structure	BN	T_F	T_B	Mean $\pm$ Std (Best)	Neurons	Params
BP (Thiele et al., 2020)	CIFARNet	×	Unknown	N.A.	(64.69%)	726K	45M
IDE (Xiao et al., 2021)	CIFARNet-F	✓	30	N.A	71.56% $\pm$ 0.31% (72.10%)	232K	14.8M
SpikeGrad (Thiele et al., 2020)	CIFARNet	×	Unknown	Unknown	(64.40%)	726K	45M
SPIDE (ours)	CIFARNet-F	×	30	100	63.57% $\pm$ 0.30%(63.91%)	232K	14.8M
SPIDE (ours)	CIFARNet-F	×	30	250	64.00% $\pm$ 0.11%(64.07%)	232K	14.8M

Table 8
Performance on CIFAR10-DVS.

Method	Model	T_F	T_B	Accuracy
Gabor-SNN (Sironi, Brambilla, Bourdis, Lagorce, & Benosman, 2018)	Gabor-SNN	N.A	N.A	24.5%
HATS (Sironi et al., 2018)	HATS	N.A	N.A	52.4%
STBP (Wu et al., 2019)	Spiking CNN (LIF, w/o NeuNorm)	40	N.A	58.1%
STBP (Wu et al., 2019)	Spiking CNN (LIF, w/ NeuNorm)	40	N.A	60.5%
Tandem learning (Wu, Chua et al., 2021)	Spiking CNN (IF)	20	N.A	58.65%
ASF-BP (Wu, Zhang et al., 2021)	Spiking CNN (IF)	Unknown	N.A	62.5%
SPIDE (ours)	Spiking CNN (IF)	30	250	60.7%

Table 9
Performance of LIF neurons on MNIST, CIFAR-10, and CIFAR-100. Results are based on 3 runs of experiments.

Dataset	Forward model	Backward model	T_F	T_B	Mean $\pm$ Std (Best)
	IF	IF	30	100	99.34% $\pm$ 0.02% (99.37%)
MNIST	LIF	IF	30	100	99.32% $\pm$ 0.04% (99.37%)
	LIF	LIF	30	100	99.34% $\pm$ 0.05% (99.39%)
	IF	IF	30	250	89.94% $\pm$ 0.17% (90.13%)
CIFAR-10	LIF	IF	30	250	89.66% $\pm$ 0.12% (89.78%)
	LIF	LIF	30	250	89.54% $\pm$ 0.14% (89.72%)
	IF	IF	30	250	64.00% $\pm$ 0.11% (64.07%)
CIFAR-100	LIF	IF	30	250	64.04% $\pm$ 0.03% (64.06%)
	LIF	LIF	30	250	63.81% $\pm$ 0.19% (63.97%)

this technique to our experiment on CIFAR-10 and the performance of the AlexNet-F structure improves from 89.61% to 90.37% and the performance of the CIFARNet-F structure improves from 89.94% to 90.85%. It shows that we can effectively leverage other techniques to improve performance.

Results with LIF neuron model. As introduced in Section 4.2, our SPIDE method is also applicable to the LIF neuron model. We conduct experiments on MNIST, CIFAR-10, and CIFAR-100 to verify the effectiveness of SPIDE with LIF neurons. Following Xiao et al. (2021), the leaky term for LIF neurons is set as $\lambda = 0.95$ for MNIST and $\lambda = 0.99$ for CIFAR-10 and CIFAR-100. The network structure for CIFAR-10 and CIFAR-100 is taken as CIFARNet-F, and other details are the same as the experiment for IF neurons. The results are shown in Table 9. It demonstrates that SPIDE is also effective for LIF neurons.

6. Conclusion

In this work, we propose the SPIDE method that generalizes the IDE method to enable purely spike-based training of SNNs with common neuron models and flexible network structures. We prove that the implicit differentiation can be solved with spikes by our coupled neurons. We also analyze the approximation error due to finite time steps and propose to adjust the reset potential of SNNs. Experiments show that we can achieve competitive performance with a small number of training time steps and sparse spikes, which demonstrates the great potential of our method for energy-efficient training of SNNs with spike-based computation. As for the limitations, SPIDE and IDE mainly focus on the condition that inputs are convergent in the context of average accumulated signals, e.g. image classification with static or neuromorphic inputs. For long-term sequential data such as

speech, we may define the final average inputs as the convergent inputs and directly apply our method, but the error flow throughout time is not carefully handled. An interesting future work is to generalize the methodology to time-varying inputs with more careful consideration of error flows and equilibriums, e.g. with certain time windows.

Declaration of competing interest

Data availability

Data will be made available on request.

Acknowledgments

Appendix A. Proof of Theorem 1

Proof. We first prove the convergence of $\boldsymbol{\beta}[t]$. Let V_u and V_u^b denote $V_{th} - u_{reset}$ and $V_{th}^b - u_{reset}^b$ respectively. Consider $\|\boldsymbol{\beta}[t+1] - \boldsymbol{\beta}[t]\|$, it satisfies:

$$
\begin{aligned}
&\|\boldsymbol{\beta}[t+1] - \boldsymbol{\beta}[t]\| \\
= &\left\| \left(\phi \left(\frac{1}{V_u^b} \left(\frac{t}{t+1} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) - \frac{1}{V_u^b} \frac{\mathbf{u}^B[t+1]}{t+1} \right) \right. \\
&\left. - \left(\phi \left(\frac{1}{V_u^b} \left(\frac{t-1}{t} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) - \frac{1}{V_u^b} \frac{\mathbf{u}^B[t]}{t} \right) \right\| \\
\leq &\left\| \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) \right. \\
&\left. - \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) \right\| \\
&+ \left\| \phi \left(\frac{1}{V_u^b} \left(\frac{t}{t+1} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) - \frac{1}{V_u^b} \frac{\mathbf{u}^B[t+1]}{t+1} \right. \\
&\left. - \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) \right\| \\
&+ \left\| \phi \left(\frac{1}{V_u^b} \left(\frac{t-1}{t} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) - \frac{1}{V_u^b} \frac{\mathbf{u}^B[t]}{t} \right. \\
&\left. - \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) \right\| \\
\leq &\left\| \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) \right. \\
&\left. - \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) \right\| \\
&+ \frac{1}{V_u^b} \left(\left\| \frac{1}{t+1} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] \right\| + \left\| \frac{\mathbf{u}^B[t+1]}{t+1} \right\| \right. \\
&\left. + \left\| \frac{1}{t} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] \right\| + \left\| \frac{\mathbf{u}^B[t]}{t} \right\| \right). \quad \text{(A.1)}
\end{aligned}
$$

As $\|(\mathbf{MW})^\top\|_2 \leq \gamma V_u V_u^b$, $\gamma < 1$, and $|\mathbf{u}_i^B[t]|$ is bounded, we have $\|\boldsymbol{\beta}[t+1]\| \leq \gamma \|\boldsymbol{\beta}[t]\| + \frac{1}{V_u} \left(\|\mathbf{g}\| + \left\| \frac{\mathbf{u}^B[t+1]}{t+1} \right\| \right) \leq \gamma \|\boldsymbol{\beta}[t]\| + c$, where c is a constant. Therefore $\|\boldsymbol{\beta}[t]\|$ is bounded. Then $\forall \epsilon > 0$, $\exists T_1$ such that when $t > T_1$, we have:

$$
\begin{aligned}
&\frac{1}{V_u^b} \left(\left\| \frac{1}{t+1} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] \right\| + \left\| \frac{\mathbf{u}^B[t+1]}{t+1} \right\| \right. \\
&\left. + \left\| \frac{1}{t} \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] \right\| + \left\| \frac{\mathbf{u}^B[t]}{t} \right\| \right) \leq \frac{\epsilon(1-\gamma)}{2}. \quad \text{(A.2)}
\end{aligned}
$$

And since $\|(\mathbf{MW})^\top\|_2 \leq \gamma V_u V_u^b$, we have:

$$
\begin{aligned}
&\left\| \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t] + \mathbf{g} \right) \right) - \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}[t-1] + \mathbf{g} \right) \right) \right\| \\
&\leq \gamma \|\boldsymbol{\beta}[t] - \boldsymbol{\beta}[t-1]\|. \quad \text{(A.3)}
\end{aligned}
$$

Therefore, when $t > T_1$, it holds that:

$$
\|\boldsymbol{\beta}[t+1] - \boldsymbol{\beta}[t]\| \leq \gamma \|\boldsymbol{\beta}[t] - \boldsymbol{\beta}[t-1]\| + \frac{\epsilon(1-\gamma)}{2}. \quad \text{(A.4)}
$$

By iterating the above inequality, we have $\|\boldsymbol{\beta}[t+1] - \boldsymbol{\beta}[t]\| \leq \gamma^{t-T_1} \|\boldsymbol{\beta}[T_1+1] - \boldsymbol{\beta}[T_1]\| + \frac{\epsilon(1-\gamma)}{2} \left(1 + \gamma + \cdots + \gamma^{t-T_1-1} \right) < \gamma^{t-T_1} \|\boldsymbol{\beta}[T_1+1] - \boldsymbol{\beta}[T_1]\| + \frac{\epsilon}{2}$. There exists T_2 such that when $t > T_1 + T_2$, $\gamma^{t-T_1} \|\boldsymbol{\beta}[T_1+1] - \boldsymbol{\beta}[T_1]\| \leq \frac{\epsilon}{2}$, and therefore $\|\boldsymbol{\beta}[t+1] - \boldsymbol{\beta}[t]\| < \epsilon$. According to Cauchy's convergence test, the sequence $\{\boldsymbol{\beta}[t]\}_{i=0}^\infty$ converges to $\boldsymbol{\beta}^*$. Considering the limit, it satisfies $\boldsymbol{\beta}^* = \phi \left(\frac{1}{V_u^b} \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g} \right) \right)$.

When $V_{th}^b - u_{reset}^b = 1$, and there exists $\lambda < 1$ such that $\|(\mathbf{MW})^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$ and $\|\mathbf{g}\|_\infty \leq 1 - \lambda$, the equation turns into $\boldsymbol{\beta}^* = \phi \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g} \right)$. We have:

$$
\begin{aligned}
\|\boldsymbol{\beta}^*\|_\infty &= \left\| \phi \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g} \right) \right\|_\infty \\
&\leq \left\| \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* \right\|_\infty + \|\mathbf{g}\|_\infty \\
&\leq \lambda \|\boldsymbol{\beta}^*\|_\infty + \|\mathbf{g}\|_\infty. \quad \text{(A.5)}
\end{aligned}
$$

Therefore, $\|\boldsymbol{\beta}^*\|_\infty \leq \frac{\|\mathbf{g}\|_\infty}{1-\lambda} \leq 1$, and we have $\left\| \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g} \right\|_\infty \leq \left\| \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* \right\|_\infty + \|\mathbf{g}\|_\infty \leq \lambda + (1-\lambda) = 1$. It means $\boldsymbol{\beta}^* = \phi \left(\frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g} \right) = \frac{1}{V_u} (\mathbf{MW})^\top \boldsymbol{\beta}^* + \mathbf{g}$.

Taking $f_\theta(\boldsymbol{\alpha}^*) = \sigma \left(\frac{1}{V_{th} - u_{reset}} (\mathbf{W}\boldsymbol{\alpha}^* + \mathbf{Fx}^* + \mathbf{b}) \right)$ (i.e. the fixed-point equation at the equilibrium state as in Section 3.2.1) explicitly into Eq. (5), the linear equation turns into $\left(\frac{1}{V_u} (\mathbf{MW})^\top - \mathbf{I} \right) \boldsymbol{\beta} + \mathbf{g} = 0$, where $V_u, \mathbf{M}, \mathbf{g}$ are previously defined. Therefore, $\boldsymbol{\beta}^*$ satisfies this equation. And since $\|(\mathbf{MW})^\top\|_2 \leq \gamma V_u$, $\gamma < 1$, the equation has the unique solution $\boldsymbol{\beta}^*$. □

Remark 1. As for the assumptions in the theorem, firstly, when $V_{th}^b - u_{reset}^b = 1$ as we will take, the assumption for the convergence is weaker than that for the convergence in the forward stage (in Section 3.2.1), because $\|(\mathbf{MW})^\top\|_2 \leq \|\mathbf{W}\|_2$ as $\mathbf{M}$ is a diagonal mask matrix. We will restrict the spectral norm of $\mathbf{W}$ following Xiao et al. (2021) to encourage the convergence of the forward stage (in Appendix C), then this backward stage would converge as well.

The assumptions for the consistency of the solution is a sufficient condition. In practice, the weight norm will be partially restricted by weight decay and our restriction on Frobenius norm (in Appendix C), as well as the diagonal mask matrix $\mathbf{M}$ which would be sparse if the forward firing events are sparse, and we

will rescale the loss so that the input $\mathbf{g}$ is in an appropriate range, as indicated in Section 4.3. Even if these assumptions are not satisfied, we can view ϕ as a kind of empirical clipping techniques to stabilize the training, as indicated in Section 4.2. The discussion is similar for the multi-layer condition (Theorem 2) in the next section.

Appendix B. Proof of Theorem 2

Proof. We first prove the convergence of $\boldsymbol{\beta}^l[t]$. Let V_u and V_u^b denote $V_{th} - u_{reset}$ and $V_{th}^b - u_{reset}^b$ respectively. Let

$$g_N^{t+1}(\boldsymbol{\beta}, \mathbf{g}, \mathbf{u}^B) = \phi\left(\frac{1}{V_u^b}\left(\frac{t}{t+1}\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\boldsymbol{\beta} + \mathbf{g}\right)\right) - \frac{1}{V_u^b}\frac{\mathbf{u}^B}{t+1},$$

$$g_l^t(\boldsymbol{\beta}, \mathbf{u}^B) = \phi\left(\frac{1}{V_u^b}\left(\frac{1}{V_u}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top\boldsymbol{\beta}\right)\right) - \frac{1}{V_u^b}\frac{\mathbf{u}^B}{t}, l = 1, \ldots, N-1,$$

$$g_N(\boldsymbol{\beta}, \mathbf{g}) = \phi\left(\frac{1}{V_u^b}(\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\boldsymbol{\beta} + \mathbf{g})\right),$$

$$g_l(\boldsymbol{\beta}) = \phi\left(\frac{1}{V_u^b}\left(\frac{1}{V_u}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top\boldsymbol{\beta}\right)\right), l = 1, \ldots, N-1. \text{ Then } \boldsymbol{\beta}^N[t+1] = g_N^{t+1}\left(g_1^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{1^B}[t]\right), \mathbf{g}, \mathbf{u}^{N^B}[t+1]\right).$$

We have:

$$\left\|\boldsymbol{\beta}^N[t+1] - \boldsymbol{\beta}^N[t]\right\|$$

$$= \left\|g_N^{t+1}\left(g_1^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{1^B}[t]\right), \mathbf{g}, \mathbf{u}^{N^B}[t+1]\right)\right.$$

$$\left. - g_N^t\left(g_1^{t-1}\left(\cdots g_{N-1}^{t-1}\left(\boldsymbol{\beta}^N[t-1], \mathbf{u}^{N-1^B}[t-1]\right)\cdots, \mathbf{u}^{1^B}[t-1]\right), \mathbf{g}, \mathbf{u}^{N^B}[t]\right)\right\|$$

$$\leq \left\|g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right), \mathbf{g}\right) - g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right), \mathbf{g}\right)\right\|$$

$$+ \left\|g_N^{t+1}\left(g_1^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{1^B}[t]\right), \mathbf{g}, \mathbf{u}^{N^B}[t+1]\right)\right.$$

$$\left. - g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right), \mathbf{g}\right)\right\|$$

$$+ \left\|g_N^t\left(g_1^{t-1}\left(\cdots g_{N-1}^{t-1}\left(\boldsymbol{\beta}^N[t-1], \mathbf{u}^{N-1^B}[t-1]\right)\cdots, \mathbf{u}^{1^B}[t-1]\right), \mathbf{g}, \mathbf{u}^{N^B}[t]\right)\right.$$

$$\left. - g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right), \mathbf{g}\right)\right\|$$

$$\leq \left\|g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right), \mathbf{g}\right) - g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right), \mathbf{g}\right)\right\|$$

$$+ \frac{1}{V_u^b}\left(\left\|\frac{1}{t+1}\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top g_1^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{1^B}[t]\right)\right\|\right.$$

$$+ \left\|\frac{1}{t}\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top g_1^{t-1}\left(\cdots g_{N-1}^{t-1}\left(\boldsymbol{\beta}^N[t-1], \mathbf{u}^{N-1^B}[t-1]\right)\cdots, \mathbf{u}^{1^B}[t-1]\right)\right\|$$

$$+ X + Y + \left\|\frac{\mathbf{u}^{N^B}[t+1]}{t+1}\right\| + \left\|\frac{\mathbf{u}^{N^B}[t]}{t}\right\|\right), \tag{B.1}$$

where $X = \left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\left(g_1^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{1^B}[t]\right) - g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right)\right)\right\|$, and $Y = \left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\left(g_1^{t-1}\left(\cdots g_{N-1}^{t-1}\left(\boldsymbol{\beta}^N[t-1], \mathbf{u}^{N-1^B}[t-1]\right)\cdots, \mathbf{u}^{1^B}[t-1]\right) - g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right)\right)\right\|$.

For the term X and Y, they are bounded by:

$$X \leq \frac{1}{V_u^b}\left(\left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\frac{1}{V_u}(\mathbf{M}^N\mathbf{F}^N)^\top\right.\right.$$

$$\times \left(g_2^t\left(\cdots g_{N-1}^t\left(\boldsymbol{\beta}^N[t], \mathbf{u}^{N-1^B}[t]\right)\cdots, \mathbf{u}^{2^B}\right)\right.$$

$$\left.\left. - g_2\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right)\right)\right\| + \left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\frac{\mathbf{u}^{1^B}[t]}{t}\right\|\right)$$

$$\leq \cdots\cdots$$

$$\leq \frac{1}{V_u^b}\left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\frac{\mathbf{u}^{1^B}[t]}{t}\right\| + \cdots$$

$$+ \frac{1}{V_u^{b^{N-1}}}\left\|\frac{1}{V_u^{N-1}}(\mathbf{M}^1\mathbf{W}^1)^\top(\mathbf{M}^N\mathbf{F}^N)^\top\cdots(\mathbf{M}^3\mathbf{F}^3)^\top\frac{\mathbf{u}^{N-1^B}[t]}{t}\right\|, \tag{B.2}$$

and Y has the same form as X by substituting t with $t-1$.

Since $\|(\mathbf{M}^1\mathbf{W}^1)^\top\|_2\|(\mathbf{M}^N\mathbf{F}^N)^\top\|_2\cdots\|(\mathbf{M}^2\mathbf{F}^2)^\top\|_2 \leq \gamma V_u^N V_u^{b^N}$, we have:

$$\left\|g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right), \mathbf{g}\right)\right.$$

$$\left. - g_N\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right), \mathbf{g}\right)\right\|$$

$$\leq \left\|\frac{1}{V_u^b}\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\left(g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t]\right)\cdots\right)\right.\right.$$

$$\left.\left. - g_1\left(\cdots g_{N-1}\left(\boldsymbol{\beta}^N[t-1]\right)\cdots\right)\right)\right\|$$

$$\leq \cdots\cdots$$

$$\leq \left\|\frac{1}{V_u^{b^N}}\frac{1}{V_u^N}(\mathbf{M}^1\mathbf{W}^1)^\top(\mathbf{M}^N\mathbf{F}^N)^\top\cdots(\mathbf{M}^2\mathbf{F}^2)^\top\left(\boldsymbol{\beta}^N[t] - \boldsymbol{\beta}^N[t-1]\right)\right\|$$

$$\leq \gamma\left\|\boldsymbol{\beta}^N[t] - \boldsymbol{\beta}^N[t-1]\right\|. \tag{B.3}$$

And since $\mathbf{u}_i^{l^B}[t]$ is bounded, then $\forall \epsilon > 0, \exists T_1$ such that when $t > T_1$, we have:

$$\left\|\boldsymbol{\beta}^N[t+1] - \boldsymbol{\beta}^N[t]\right\| \leq \gamma\left\|\boldsymbol{\beta}^N[t] - \boldsymbol{\beta}^N[t-1]\right\| + \frac{\epsilon(1-\gamma)}{2}. \tag{B.4}$$

Then $\|\boldsymbol{\beta}^N[t+1] - \boldsymbol{\beta}^N[t]\| < \gamma^{t-T_1}\|\boldsymbol{\beta}^N[T_1+1] - \boldsymbol{\beta}^N[T_1]\| + \frac{\epsilon}{2}$, and there exists T_2 such that when $t > T_1 + T_2$, $\|\boldsymbol{\beta}^N[t+1] - \boldsymbol{\beta}^N[t]\| < \epsilon$. According to Cauchy's convergence test, $\boldsymbol{\beta}^N[t]$ converges to $\boldsymbol{\beta}^{N*}$, which satisfies $\boldsymbol{\beta}^{N*} = g_N\left(g_1 \circ \cdots \circ g_{N-1}(\boldsymbol{\beta}^{N*}), \mathbf{g}\right)$. Considering the limit, $\boldsymbol{\beta}^l[t]$ converges to $\boldsymbol{\beta}^{l*}$, which satisfies $\boldsymbol{\beta}^{l*} = g_l(\boldsymbol{\beta}^{l+1*})$.

When $V_{th}^b - u_{reset}^b = 1$, and there exists $\lambda < 1$ such that $\|(\mathbf{M}^1\mathbf{W}^1)^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$, $\|(\mathbf{M}^l\mathbf{F}^l)^\top\|_\infty \leq \lambda(V_{th} - u_{reset})$, $l = 2, \ldots, N$ and $\|\mathbf{g}\|_\infty \leq 1 - \lambda^N$, we have:

$$
\begin{aligned}
\left\|\boldsymbol{\beta}^{N*}\right\|_\infty &= \left\|g_N\left(g_1 \circ \cdots \circ g_{N-1}(\boldsymbol{\beta}^{N*}), \mathbf{g}\right)\right\|_\infty \\
&\leq \left\|\frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top g_1 \circ \cdots \circ g_{N-1}(\boldsymbol{\beta}^{N*})\right\|_\infty + \|\mathbf{g}\|_\infty \\
&\leq \lambda\left\|g_1 \circ \cdots \circ g_{N-1}(\boldsymbol{\beta}^{N*})\right\|_\infty + \|\mathbf{g}\|_\infty \\
&\leq \cdots\cdots \leq \lambda^N\left\|\boldsymbol{\beta}^{N*}\right\|_\infty + \|\mathbf{g}\|_\infty
\end{aligned}
\tag{B.5}
$$

Therefore, $\left\|\boldsymbol{\beta}^{N*}\right\|_\infty \leq \frac{\|\mathbf{g}\|_\infty}{1-\lambda^N} \leq 1$, and $\left\|\tilde{g}_{N-1}(\boldsymbol{\beta}^{N*})\right\|_\infty \leq \lambda\left\|\boldsymbol{\beta}^{N*}\right\|_\infty \leq \lambda, \ldots, \left\|\tilde{g}_1 \circ \cdots \circ \tilde{g}_{N-1}(\boldsymbol{\beta}^{N*})\right\|_\infty \leq \lambda^{N-1}$, $\left\|\tilde{g}_N\left(\tilde{g}_1 \circ \cdots \circ \tilde{g}_{N-1}(\boldsymbol{\beta}^{N*}), \mathbf{g}\right)\right\|_\infty \leq \lambda^N + (1 - \lambda^N) = 1$, where $\tilde{g}_N(\boldsymbol{\beta}, \mathbf{g}) = \frac{1}{V_u}(\mathbf{M}^1\mathbf{W}^1)^\top\boldsymbol{\beta} + \mathbf{g}$, $\tilde{g}_l(\boldsymbol{\beta}) = \frac{1}{V_u}(\mathbf{M}^{l+1}\mathbf{F}^{l+1})^\top\boldsymbol{\beta}$, $l = 1, \ldots, N - 1$, (i.e. $\tilde{g}_l$ is g_l without the function ϕ). It means $\boldsymbol{\beta}^{N*} = g_N\left(g_1 \circ \cdots \circ g_{N-1}(\boldsymbol{\beta}^{N*}), \mathbf{g}\right) = \tilde{g}_N\left(\tilde{g}_1 \circ \cdots \circ \tilde{g}_{N-1}(\boldsymbol{\beta}^{N*}), \mathbf{g}\right)$ and $\boldsymbol{\beta}^{l*} = g_l(\boldsymbol{\beta}^{l+1*}) = \tilde{g}_l(\boldsymbol{\beta}^{l+1*})$.

Taking $\boldsymbol{\alpha}^{1*} = f_1\left(f_N \circ \cdots \circ f_2(\boldsymbol{\alpha}^{1*}), \mathbf{x}^*\right)$ and $\boldsymbol{\alpha}^{l+1*} = f_{l+1}(\boldsymbol{\alpha}^{l*})$ (i.e. the fixed-point equation at the equilibrium state as in Section 3.2.1) explicitly into Eq. (5), the linear equation turns into $\tilde{g}_1 \circ \cdots \circ \tilde{g}_{N-1}(\boldsymbol{\beta}) - \boldsymbol{\beta} + \mathbf{g} = 0$. Therefore, $\boldsymbol{\beta}^{N*}$ satisfies this equation. And since $\|(\mathbf{M}^1\mathbf{W}^1)^\top\|_2\|(\mathbf{M}^N\mathbf{F}^N)^\top\|_2 \cdots \|(\mathbf{M}^2\mathbf{F}^2)^\top\|_2 \leq \gamma V_u^N, \gamma < 1$, the equation has the unique solution $\boldsymbol{\beta}^{N*}$. Further, because $\tilde{g}_l(\boldsymbol{\beta}) = \left(\frac{\partial h_{l+1}(\boldsymbol{\alpha}^{N*})}{\partial h_l(\boldsymbol{\alpha}^{N*})}\right)^\top\boldsymbol{\beta}$, where $h_l(\boldsymbol{\alpha}^{N*}) = f_l \circ \cdots \circ f_2\left(f_1(\boldsymbol{\alpha}^{N*}, \mathbf{x}^*)\right), l = N, \ldots, 1$, we have $\boldsymbol{\beta}^{l*} = \left(\frac{\partial h_N(\boldsymbol{\alpha}^{N*})}{\partial h_l(\boldsymbol{\alpha}^{N*})}\right)^\top\boldsymbol{\beta}^{N*}, l = N - 1, \ldots, 1$. $\square$

Appendix C. Training details

C.1. Dropout

Dropout is a commonly used technique to prevent over-fitting, and we follow Bai et al. (2019, 2020) and Xiao et al. (2021) to leverage variational dropout, i.e. the dropout of each layer is the same at different time steps. Since applying dropout on the output of neurons is a linear operation with a mask and scaling factor, it can be integrated into the weight matrix without affecting the conclusions of convergence. The detailed computation with dropout is also illustrated in the pseudocode in Section 4.4.

C.2. Restriction on weight norm

As indicated in the theorems, a sufficient condition for the convergence to equilibrium states in both forward and backward stages is the restriction on the weight spectral norm. Xiao et al. (2021) leverages re-parameterization to restrict the spectral norm, i.e. they re-parameterize $\mathbf{W}$ as $\mathbf{W} = \alpha\frac{\mathbf{W}}{\|\mathbf{W}\|_2}$, where $\|\mathbf{W}\|_2$ is computed as the implementation of Spectral Normalization and α is a learnable parameter to be clipped in the range of $[-c, c]$ (c is a constant). However, the computation of spectral norm and re-parameterization may be unfriendly to neuromorphic computation. We adjust it for a more friendly calculation as follows.

First, the spectral norm is upper-bounded by the Frobenius norm: $\|\mathbf{W}\|_2 \leq \|\mathbf{W}\|_F$. We can alternatively restrict the Frobenius norm which is easier to compute. Further, considering that connection weights may not be easy for readout compared with neuron outputs, we can approximate $\|\mathbf{W}\|_F$ by $\|\mathbf{W}\|_F = \sqrt{\text{tr}(\mathbf{W}\mathbf{W}^\top)} =$

$\sqrt{\mathbb{E}_{\boldsymbol{\epsilon}\in\mathcal{N}(0,\mathbf{I}_d)}\left[\|\boldsymbol{\epsilon}^\top\mathbf{W}\|_2^2\right]}$, according to the Hutchinson estimator (Hutchinson, 1989). It can be viewed as source neurons outputting noises and target neurons accumulating signals to estimate the Frobenius norm. We will estimate the norm based on the Monte-Carlo estimation (we will take 64 samples, which is similarly adopted by Bai, Koltun, and Kolter (2021) to estimate the norm of their Jacobian matrix. Then based on the estimation, we will restrict $\mathbf{W}$ as $\mathbf{W} = \alpha\frac{\mathbf{W}}{\|\mathbf{W}\|_F}$ where $\alpha = \min(c, \|\mathbf{W}\|_F)$, c is a constant for norm range. This estimation and calculation may correspond to large amounts of noises in our brains, and a feedback inhibition on connection weights based on neuron outputs.

Following Xiao et al. (2021), we only restrict the norm of feedback connection weight $\mathbf{W}^1$ for the multi-layer structure, which works well in practice.

C.3. Other details

Details on MNIST, CIFAR-10, and CIFAR-100. For SNN models with feedback structure, we set $V_{th} = 1, u_{reset} = -1$ in the forward stage to form an equivalent equilibrium state as Xiao et al. (2021). The constant for restriction in Appendix C.2 is $c = 2$. Following Xiao et al. (2021), we train models by SGD with momentum for 100 epochs. The momentum is 0.9, the batch size is 128, and the initial learning rate is 0.05. For MNIST, the learning rate is decayed by 0.1 every 30 epochs, while for CIFAR-10 and CIFAR-100, it is decayed by 0.1 at the 50th and 75th epoch. We apply linear warmup for the learning rate in the first 400 iterations for CIFAR-10 and CIFAR-100. We apply the weight decay with 5×10^{-4} and variational dropout with the rate 0.2 for AlexNet-F and 0.25 for CIFARNet-F. The initialization of weights follows Wu et al. (2018), i.e. we sample weights from the standard uniform distribution and normalize them on each output dimension. The scale for the loss function (as in Section 4.3) is 100 for MNIST, 400 for CIFAR-10, and 500 for CIFAR-100.

For SNN models with degraded feedforward structure, our hyperparameters mostly follow Thiele et al. (2020), i.e. we set $V_{th} = 0.5, u_{reset} = -0.5$, train models by SGD with momentum 0.9 for 60 epochs, set batch size as 128, and the initial learning rate as 0.1 which is decayed by 0.1 every 20 epochs, and apply the variational dropout only on the first fully-connected layer with rate 0.5.

The notations for our structures mean: '64C5' represents a convolution operation with 64 output channels and kernel size 5, 's' after '64C5' means convolution with stride 2 (which downscales $2\times$) while 'u' after that means a transposed convolution to upscale $2\times$, 'P2' means average pooling with size 2, and 'F' means feedback layers. The network structures for CIFAR-10 and CIFAR-100 are:

AlexNet (Wu et al., 2019): 96C3-256C3-P2-384C3-P2-384C3-256C3-1024-1024,

AlexNet-F (Xiao et al., 2021): 96C3s-256C3-384C3s-384C3-256C3 (F96C3u),

CIFARNet (Wu et al., 2019): 128C3-256C3-P2-512C3-P2-1024C3-512C3-1024-512,

CIFARNet-F (Xiao et al., 2021): 128C3s-256C3-512C3s-1024C3-512C3 (F128C3u).

Details on CIFAR10-DVS. The CIFAR10-DVS dataset is the neuromorphic version of the CIFAR-10 dataset converted by a Dynamic Vision Sensor (DVS), which is composed of 10,000 samples, one-sixth of the original CIFAR-10. It consists of spike trains with two channels corresponding to ON- and OFF-event spikes. The pixel dimension is expanded to 128×128. Following the common practice, we split the dataset into 9000 training samples and

1000 testing samples. As for the data pre-processing, we reduce the time resolution by accumulating the spike events (Fang, Yu, Chen, Masquelier et al., 2021) into 30 time steps, and we reduce the spatial resolution into 48×48 by interpolation. We apply the same random crop augmentation as CIFAR-10 to the input data. We leverage the network structure: 512C9s (F512C5). We train the model by SGD with momentum for 70 epochs. The momentum is 0.9, the batch size is 128, the weight-decay is 5×10^{-4}, and the initial learning rate is 0.05 which is decayed by 0.1 at the 50th epoch. No dropout is applied. The initialization of weights follows the widely used Kaiming initialization. The constant for restriction in Appendix C.2 is $c = 10$ due to the large channel size, and the scale for the loss function as well as the firing thresholds and reset potentials are the same as the CIFAR-10 experiment.

We simulate the computation on commonly used computational units. The code implementation is based on the PyTorch framework (Paszke et al., 2019), and experiments are carried out on one NVIDIA GeForce RTX 3090 GPU.

References

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). TrueNorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, *34*(10), 1537–1557.

Almeida, L. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *International conference on neural networks*.

Bai, S., Kolter, J. Z., & Koltun, V. (2019). Deep equilibrium models. In *Advances in neural information processing systems*.

Bai, S., Koltun, V., & Kolter, J. Z. (2020). Multiscale deep equilibrium models. In *Advances in neural information processing systems*.

Bai, S., Koltun, V., & Kolter, Z. (2021). Stabilizing equilibrium models by Jacobian regularization. In *International conference on machine learning*.

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

Bohte, S. M., Kok, J. N., & La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1–4), 17–37.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*.

Brock, A., De, S., & Smith, S. L. (2021). Characterizing signal propagation to close the performance gap in unnormalized resnets. In *International conference on learning representations*.

Brock, A., De, S., Smith, S. L., & Simonyan, K. (2021). High-performance large-scale image recognition without normalization. In *International conference on machine learning*.

Bu, T., Fang, W., Ding, J., Dai, P., Yu, Z., & Huang, T. (2022). Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International conference on learning representations*.

Chen, J., Gai, Y., Yao, Z., Mahoney, M. W., & Gonzalez, J. E. (2020). A statistical framework for low-bitwidth training of deep neural networks. In *Advances in neural information processing systems*.

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

Deng, S., & Gu, S. (2021). Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International conference on learning representations*.

Deng, S., Li, Y., Zhang, S., & Gu, S. (2022). Temporal efficient training of spiking neural network via gradient re-weighting. In *International conference on learning representations*.

Detorakis, G., Bartley, T., & Neftci, E. (2019). Contrastive hebbian learning with random feedback weights. *Neural Netw.*, *114*, 1–14.

Diehl, P. U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, *9*, 99.

Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., & Tian, Y. (2021). Deep residual learning in spiking neural networks. In *Advances in neural information processing systems*.

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

Guerguiev, J., Lillicrap, T. P., & Richards, B. A. (2017). Towards deep learning with segregated dendrites. *eLife*, *6*, Article e22901.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.*, *79*(8), 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci.*, *81*(10), 3088–3092.

Hunsberger, E., & Eliasmith, C. (2015). Spiking deep networks with LIF neurons. arXiv preprint arXiv:1510.08829.

Hutchinson, M. F. (1989). A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Comm. Statist. Simulation Comput.*, *18*(3), 1059–1076.

Jin, Y., Zhang, W., & Li, P. (2018). Hybrid macro/micro level backpropagation for training deep spiking neural networks. In *Advances in neural information processing systems*.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., & Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.*, *99*, 56–67.

Kim, J., Kim, K., & Kim, J.-J. (2020). Unifying activation-and timing-based learning rules for spiking neural networks. In *Advances in neural information processing systems*.

Kim, Y., Li, Y., Park, H., Venkatesha, Y., & Panda, P. (2022). Neural architecture search for spiking neural networks. arXiv preprint arXiv:2201.10355.

Kim, S., Park, S., Na, B., & Yoon, S. (2020). Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images: Technical report*, University of Toronto.

Kubilius, J., Schrimpf, M., Kar, K., Rajalingham, R., Hong, H., Majaj, N., et al. (2019). Brain-like object recognition with high-performing shallow recurrent ANNs. In *Advances in neural information processing systems*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, *86*(11), 2278–2324.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.*, *10*, 508.

Legenstein, R., Pecevski, D., & Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), Article e1000180.

Li, Y., Deng, S., Dong, X., Gong, R., & Gu, S. (2021). A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration. In *International conference on machine learning*.

Li, Y., Guo, Y., Zhang, S., Deng, S., Hai, Y., & Gu, S. (2021). Differentiable spike: Rethinking gradient-descent for training spiking neural networks. In *Advances in neural information processing systems*.

Li, H., Liu, H., Ji, X., Li, G., & Shi, L. (2017). Cifar10-dvs: an event-stream dataset for object classification. *Front. Neurosci.*, *11*, 309.

Li, Q., & Pehlevan, C. (2020). Minimax dynamics of optimally balanced spiking networks of excitatory and inhibitory neurons. In *Advances in neural information processing systems*.

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

Mancoo, A., Keemink, S., & Machens, C. K. (2020). Understanding spiking networks through convex optimization. In *Advances in neural information processing systems*.

Martin, E., Ernoult, M., Laydevant, J., Li, S., Querlioz, D., Petrisor, T., et al. (2021). Eqspike: spike-driven equilibrium propagation for neuromorphic implementations. *iScience*, *24*(3), Article 102222.

Meng, Q., Xiao, M., Yan, S., Wang, Y., Lin, Z., & Luo, Z.-Q. (2022). Training high-performance low-latency spiking neural networks by differentiation on spike representation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Meng, Q., Yan, S., Xiao, M., Wang, Y., Lin, Z., & Luo, Z.-Q. (2022). Training much deeper spiking neural networks with a small number of time-steps. *Neural Netw.*, *153*, 254–268.

Mesnard, T., Gerstner, W., & Brea, J. (2016). Towards deep learning with spiking neurons in energy based models with contrastive hebbian plasticity. arXiv preprint arXiv:1612.03214.

Neftci, E. O., Augustine, C., Paul, S., & Detorakis, G. (2017). Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Front. Neurosci.*, *11*, 324.

Neftci, E. O., Mostafa, H., & Zenke, F. (2019). Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.*, *36*(6), 51–63.

Nøkland, A. (2016). Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*.

O'Connor, P., Gavves, E., & Welling, M. (2019). Training a spiking neural network with equilibrium propagation. In *The 22nd international conference on artificial intelligence and statistics*.

O'Connor, P., & Welling, M. (2016). Deep spiking networks. arXiv preprint arXiv:1602.08323.