ACCELERATING TRAINING OF DEEP SPIKING NEURAL NETWORKS WITH PARAMETER INITIALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite that spiking neural networks (SNNs) show strong advantages in information encoding, power consuming, and computational capability, the underdevelopment of supervised learning algorithms is still a hindrance for training SNN. Our consideration is that proper weight initialization is a pivotal issue for efficient SNN training. It greatly influences gradient generating with the method of back-propagation through time at the initial training stage. Focusing on the properties of spiking neurons, we first derive the asymptotic formula of their response curve approximating the actual neuron response distribution. Then, we propose an initialization method obtained from the slant asymptote to overcome gradient vanishing. Finally, experiments with different coding schemes on classification tasks show that our method can effectively improve training speed and the final model accuracy compared with traditional deep learning initialization methods and existing SNN initialization methods. Further validation on different neuron types and training hyper-parameters has shown comparably good versatility and superiority over the other methods. Some suggestions are given to SNN training based on the analyses.

1 INTRODUCTION

Deep learning has currently contributed to outstanding performances in various tasks (Hinton et al., 2012; Hosu & Rebedea, 2016; He et al., 2016; LeCun et al., 2015). Most state-of-the-art models are based on analog neurons, which take analog values as input and produce output via activation functions (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015). Each analog value conveys activation, thus inference of analog neural networks (ANNs) consists of massive matrix manipulations. To overcome this, spiking neural networks (SNNs) is attracting more and more researchers (Maass, 1997; Maass & Markram, 2004; Yang et al., 2019). Spiking neurons in SNN emit sparse 0,1 spike when activated. The forward computation of SNN is event-driven in neuromorphic hardware including TrueNorth, SpiNNaker, Loihi, and so on (DeBole et al., 2019; Painkras et al., 2013; Davies et al., 2018). Compared with existing ANNs, SNNs are considered to encode more spatio-temporal patterns and have better power efficiency. Hence, implementations of SNNs are highly appealing in power-limited scenes (Ding et al., 2021). Besides, spiking neural networks have exhibited their computational capability as powerful as conventional artificial neural networks (Lee et al., 2016).

Despite SNNs' powerful capability, SNNs have not yet been used in advantageous scenarios (such as real-time ones). One reason can be attributed to the lack of efficient supervised learning algorithms. Discrete spikes disable the direct back-propagation (BP) training. Recent proposed supervised algorithms mainly focused on two methodologies. One is to convert a pre-trained ANN to an SNN using training data response (Diehl et al., 2015; Sengupta et al., 2019) or better activation design (Deng & Gu, 2021; Ho & Chang, 2020). The aim of conversion methods is to maintain comparable ANN accuracy and not seriously increase the inference timesteps. The other is to BP through time (BPTT) with surrogate functions (Lee et al., 2016; Zheng et al., 2021; Neftci et al., 2019). However, some literature has reported the limitation that only training with shallow architectures is feasible (Wu et al., 2018). Therefore, BPTT training still cannot be regarded as a mature SNN training algorithm.

We ascribe the immaturity of BPTT training to the gradients. One common reason is gradient vanishing and explosion. It is natural since SNNs can be regarded as a type of Recurrent neural networks (RNNs) (He et al., 2020; Fang et al., 2020). In the wisdom of RNN methodology, researchers man-

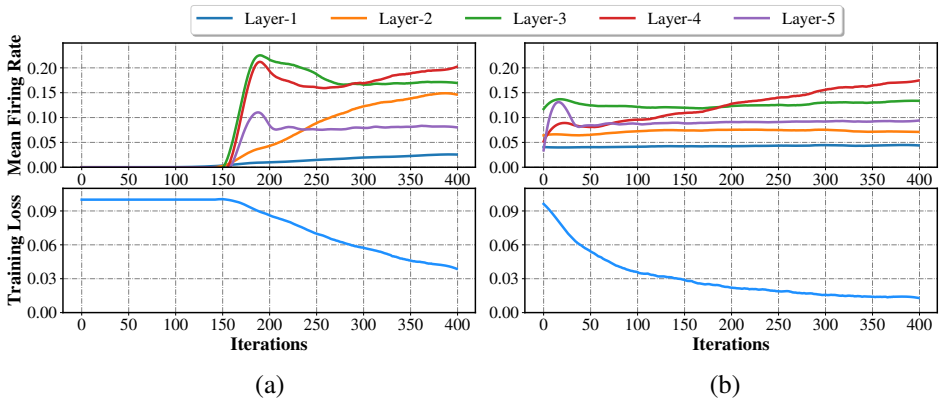


Figure 1: Loss curves and mean firing rate curves of a 5-layered SNN, initialized by methods proposed by Glorot & Bengio (2010) (a) and us (b). Figure (a) is an example of inadequate initialization. All losses start to decrease until the firing rates increase. With proper initialization, firing rates increase early and loss decreases lower and faster.

aged to give a simple solution by rethinking weight initialization (Le et al., 2015). Gradients in SNNs are determined by nonlinear surrogate functions, in which gradients rarely equal 1. In this case, gradient vanishing and explosion will be more serious than RNNs, which prevents deep SNNs from reliable training. The success of RNN training inspires thinking about whether there will be also a simple solution in SNN. Another reason is early gradient generating, which is rarely mentioned in previous literature. Spiking neurons emit spikes when membrane potentials exceed the threshold. Most surrogate functions produce gradients in the neighborhood of the threshold. Some produce gradients only when spikes are fired (Lee et al., 2020a). Thus, the flow of gradients heavily relies on the number of firing neurons and will influence the optimization of the network. A small experiment can help explain how early gradient generating affects training. As shown in Fig. 1, with inadequate weight initialization method, neurons take in too small inputs and cannot fire initially, leading to the stall of training because gradients fail to be produced. Until some neurons fire, the loss begins to decrease. And neurons will have to bear the latter uncoordinated training schedule. In general, good weight initialization is also important for efficient SNN training.

At present, SNN training initialization mainly follows and tunes ANN methods without proper analysis. But unlike ReLU in ANN, spiking neurons respond very differently, recent improvements of models have rarely focused on the properties of spiking neurons. In this paper, we investigate the theoretical response of spiking neurons to get a better SNN initialization strategy. The main contributions of this paper are summarized as follows:

- We use iterative systems to model first-order integrate-and-fire neurons and theoretically investigate the response curve. A slant asymptote is derived to approximate the actual neuron response.
- We examine the difficulty of training deep SNNs and obtain an initialization with the slant asymptote. The proposed initialization uses adequate scaled weights and non-zero biases to enable early gradient training.
- Experiments with rate coding and event coding on classification tasks show that our method can effectively improve the model training speed and final accuracy. In addition, empirical analyses suggest that our asymptote initialization has versatility and superiority over vast training hyper-parameters such as neuron types, surrogate functions, and optimizers.

2 Theoretical analysis of spiking neural networks

In this section, a first-order linear integrate-and-fire model is first formulated as a type of iterative system. Based on this, we analyze the nonlinear dynamics of this type of neuron and obtain the theoretical asymptote for the neuron response.

Table 1: Parameter setting samples of first-order linear integrate-and-fire models expressed in Eq. 2.

Model	k	λ
LIF model in Eq. 1(a) (Abbott, 1999)	$1-1/\tau$	R/τ
IF model in Eq. 1(b) (Abbott, 1999)	1	$1/C$
Diehl et al. (2015); Sengupta et al. (2019)	1	1
Wu et al. (2018); Zheng et al. (2021)	τ_{decay}	1
Panda et al. (2020)	α	1

2.1 Spiking neurons and surrogate functions

With the development of neuroscience, many advanced neuron models have been proposed (Hodgkin & Huxley, 1952; Gerstner et al., 2014), commonly in the form of differential systems. However, in the field of computing, analyses are often made on first order differential models because of computational efficiency. Typically, leaky and non-leaky integrate-and-fire (LIF & IF) model can be formulated by the dynamic of membrane potential $\boldsymbol{v}$ as Eq. 1(a) and (b).

$$\begin{aligned} \tau\frac{d\boldsymbol{v}(t)}{dt} &= -(\boldsymbol{v}(t)-v_{reset})+R\boldsymbol{i}(t) && (a)\ \tau \text{ is time constant.} \\ \frac{d\boldsymbol{i}(t)}{dt} &= \frac{1}{C}\boldsymbol{i}(t) && (b) \end{aligned} \tag{1}$$

By utilizing the Euler method, we can convert the dynamic systems to an iterative expression which computers prefer (Eq. 2). Note that $\boldsymbol{v}_{reset}$ is usually 0 in literature, so it is neglected in latter analyses. When the element of $\boldsymbol{v}^t$ is larger than the firing threshold θ, the corresponding neuron will emit a spike 1, and membrane potential will be reset to 0. Without loss of generality, we use the following equations as the iterative systems for first-order integrate-and-fire neurons.

$$\begin{aligned} \boldsymbol{u}^t &= k\boldsymbol{v}^t+\lambda\boldsymbol{i}^t \\ \boldsymbol{o}^t &= \boldsymbol{u}^t>\theta \\ \boldsymbol{v}^{t+1} &= \boldsymbol{u}^t(1-\boldsymbol{o}^t) \end{aligned} \tag{2}$$

In Eq. 2, k and λ are hyperparameters that control the neuron type and firing pattern ($k\in(0,1]$). When k is set to 0, the membrane potential cancels the iteration of time and the network is not an standard SNN. Many neuron models used in existing SNN training literature can be seen as different settings of the two parameters. Table 1 illustrates some of the settings.

Within Eq. 2, $\boldsymbol{o}^t=\boldsymbol{u}^t>\theta$ is the non-differential part. It can also be rewritten by $\boldsymbol{o}^t=H(\boldsymbol{u}^t-\theta)$ where $H(\cdot)$ is Heaviside step function. $H(x)=0$ when $x\leq 0$; $H(x)=1$ when $x>0$. Heaviside function has 0 gradient in almost everywhere. To cope with non-differential problem, surrogate function methods first appeared in network quantization (Mairal, 2013; Hubara et al., 2016). To this end, four surrogate functions (Sigmoid, Tanh, Hardtanh, Arctan) are presented in this paper with parameter a, which are listed in Table A3. They are the shifted and scaled version of the original functions, as shown in Fig. A6 in the appendix. Adjusting parameter a can rescale the output gradient. The input use Heaviside function to perform forward pass, and get the backward gradient using surrogate functions. The gradient function are typically symmetric w.r.t. zero input. One exception is proposed by Lee et al. (2020a) where only the production of spikes can trigger non-zero gradients.

2.2 Response of spiking neurons and asymptotes

After the fact that early gradient generating affects SNN training has been noticed in Sec. 1, a very urgent problem is to obtain the relationship between the expected input current and the response of the output firing rate.

However, it is not easy to establish the relationship between current input and output on the iterative equation. Considering that the difference between the iterative equation and the differential equation is small when $stepsize=1$, and the differential equation can use integral to get the response curve. Theorem 1 gives the response of first-order integrate-and-fire neurons.

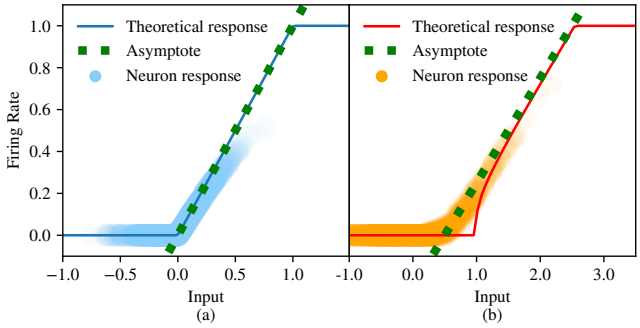


Figure 2: Theoretical response curve after integration (solid lines), slant asymptote (dashed lines), and actual neuron response (circular dots) of the first-order linear integrate-and-fire model. (a) $k = 1$, (b) $k \in (0, 1)$.

Theorem 1. *For first-order integrate-and-fire neurons expressed in Eq. 2 ($u^t = kv^t + \lambda i^t, k \in (0, 1]$), the model takes constant current $i^t = i$ as input and output spikes. Denote the discrete output firing rate w.r.t. i as $f = \frac{T}{N}$, where T is the total inference time and N is the spike count. Then the input-output response curve (f-i curve) is estimated by:*

$$
\begin{aligned}
&\textit{If } k = 1: \\
&\quad f = \begin{cases} 0, & i \leq 0, \\ \frac{\lambda}{\theta} i, & 0 < i < \frac{\theta}{\lambda}, \\ 1, & i \geq \frac{\theta}{\lambda}. \end{cases} \\
&\textit{If } k \in (0, 1): \\
&\quad f = \begin{cases} 0, & i \leq 0, \\ (k-1)/\ln(1 + \frac{(k-1)\theta}{\lambda i}), & 0 < i < \frac{\theta}{\lambda} \cdot \frac{k-1}{e^{k-1}-1}, \\ 1, & i \geq \frac{\theta}{\lambda} \cdot \frac{k-1}{e^{k-1}-1}. \end{cases}
\end{aligned}
\tag{3}
$$

Solid lines in Fig. 2 represent the response curves when $k = 1$ and $k \in (0, 1)$ in Theorem 1. Fig. 2 also make scatter plots of the actual response of spiking neurons given corresponding parameters (marked with circular dots). It can be seen that the actual neuron response is almost distributed on a continuous line. Consider the region where $f \in (0, 1)$. When $k \in (0, 1)$, it can be intuitively found that the f-i curve, which has a non-differential point, seems to exist a slant asymptote. Theorem 2 manifests the existence of the slant asymptote and gives us a uniform expression of it. In Fig. 2, this asymptote approximately crosses the actual neuron response distribution. In this case, the slant asymptote of f-i can better describe the relationship between the expected input current and the output firing rate.

Theorem 2. *For first-order integrate-and-fire neurons expressed in Eq. 2 ($u^t = kv^t + \lambda i^t, k \in (0, 1]$), the slant asymptote of f-i satisfying $f \in (0, 1)$ exists and can be uniformed as:*

$$
f_a = \frac{\lambda}{\theta} i + \frac{k-1}{2}. \tag{4}
$$

3 Weight Initialization for SNN Training

The spiking neurons' response is estimated by a piece-wise linear function in the previous analysis. So gradient initialization method should also be quite different from that of ANNs. In this section, a robust initialization method is proposed specialized for SNNs that enabling gradient update even when training deep SNNs.

When the neurons respond in a suitable region, a responsible number of spikes can drive a network to be trained better. To better illustrate this, given an L-layered SNN, we annotate the input current, membrane potential, intermediate variable, and spike output of the lth layer as $\boldsymbol{i}^{l,t}, \boldsymbol{v}^{l,t}, \boldsymbol{u}^{l,t}, \boldsymbol{o}^{l,t}$ ($l = 1, 2, \cdots, L$). Variables for one layer neurons satisfy Eq. 2. $\boldsymbol{i}^{l,t}$ is the weighted input of the previous layer's spike output ($\boldsymbol{i}^{l,t} = \boldsymbol{W}^l \boldsymbol{o}^{l-1,t} + \boldsymbol{b}^l$), and the first layer takes the encoded spike $\boldsymbol{o}^{0,t}$. Literature (Wu et al., 2018; Fang et al., 2020) has included detailed

derivation of gradient when backpropagating loss ε through time T. The gradients w.r.t. the weights of the lth layer can be expressed in Eq. 5. It implies that the gradient of the weights is, directly and indirectly, related to the output spike of the previous layer. Inappropriate $\boldsymbol{o}^{l-1,t}$, like zero output, will lead to a bad gradient.

$$\frac{\partial \varepsilon}{\partial \boldsymbol{W}^l} = \sum_{t=1}^{T} \frac{\partial \varepsilon}{\partial \boldsymbol{u}^{l,t}} \frac{\partial \boldsymbol{u}^{l,t}}{\partial \boldsymbol{i}^{l,t}} \frac{\partial \boldsymbol{i}^{l,t}}{\partial \boldsymbol{W}^l} = \lambda \sum_{t=1}^{T} \frac{\partial \varepsilon}{\partial \boldsymbol{u}^{l,t}} (\boldsymbol{o}^{l-1,t})^{\mathrm{T}} \tag{5}$$

So the focus is how to produce proper amount of spike in the forward propagation. Thankfully, the clipped version of the slant asymptote can replace the analysis of complex iterative functions and focus on the scale of parameters:

$$\begin{aligned} \hat{\boldsymbol{i}}^l &= \boldsymbol{W}^l \hat{\boldsymbol{o}}^{l-1} + \boldsymbol{b}^l \\ \hat{\boldsymbol{o}}^l &= \mathrm{clip}(\frac{\lambda}{\theta} \hat{\boldsymbol{i}}^l + \frac{k-1}{2}, 0, 1) \end{aligned} \tag{6}$$

where $\hat{\boldsymbol{i}}^i$ and $\hat{\boldsymbol{o}}^i$ are the expected current and firing rate vector of the ith layer with n_i neurons. $\boldsymbol{W}^i$ and $\boldsymbol{b}^i$ are the corresponding parameters. Note that $\boldsymbol{W}^i$ is a $n_i \times n_{i-1}$ matrix. λ, k and θ represent the neuron parameters.

At first, all weights in SNNs are initialized to be independent and identically distributed (i.i.d.). Assume that the input current variances are the same. Then we have:

$$\mathrm{Var}(\hat{\mathrm{i}}^l) = n_{l-1} \mathrm{Var}(\mathbf{W}^l \hat{\mathrm{o}}^{l-1}) \tag{7}$$

where $\hat{\mathrm{i}}^l$, $\mathbf{W}^l$ and $\hat{\mathrm{o}}^{l-1}$ denote random variables of elements in $\hat{\boldsymbol{i}}^l$, $\boldsymbol{W}^l$ and $\hat{\boldsymbol{o}}^{l-1}$ respectively. Considering $\mathrm{Var}(\mathbf{W}^l \hat{\mathrm{o}}^{l-1})$, $\mathrm{Var}(\mathbf{W}^l \hat{\mathrm{o}}^{l-1}) = \mathrm{Var}(\mathbf{W}^l)\mathbb{E}[(\hat{\mathrm{o}}^{l-1})^2] + \mathrm{Var}(\hat{\mathrm{o}}^{l-1})\mathbb{E}[\mathbf{W}^l]^2$, let $\mathbf{W}_l$ is zero mean and then,

$$\mathrm{Var}(\hat{\mathrm{i}}^l) = n_{l-1} \mathrm{Var}(\mathbf{W}^l) \mathbb{E}[(\hat{\mathrm{o}}^{l-1})^2]. \tag{8}$$

To solve $\mathbb{E}[(\hat{\mathrm{o}}^{l-1})^2]$, a consideration is that we want initial parameters to trigger spikes, as mentioned in Sec. 1. In function $\hat{o} = \mathrm{clip}(\frac{\lambda}{\theta}\hat{i} + \frac{k-1}{2}, 0, 1)$, $i = \frac{\theta(1-k)}{2\lambda}$ is the point that connects the non-spiking region and linear response region. When initializing $\boldsymbol{b}^l$ as $\frac{\theta(1-k)}{2\lambda}$ ($l = 1, 2, \cdots, L$), zero-centered $\boldsymbol{W}^l \times \hat{\boldsymbol{o}}^{l-1}$ produces symmetric distribution around zero. This can help the network to produce gradients at the initial stage of training.

With $\boldsymbol{b}^l$ initialized as $\frac{\theta(1-k)}{2\lambda}$, $\mathbb{E}[(\hat{\mathrm{o}}^{l-1})^2] = \frac{1}{2}\mathrm{Var}(\frac{\lambda}{\theta}\hat{\mathrm{i}}^{l-1}) = \frac{1}{2}\frac{\lambda^2}{\theta^2}\mathrm{Var}(\hat{\mathrm{i}}^{l-1})$. By plugging $\mathbb{E}[(\hat{\mathrm{o}}^{l-1})^2]$ into Eq. 8, we can get the relationship of variance of $\hat{\mathrm{i}}^l$ and $\hat{\mathrm{i}}^{l-1}$:

$$\mathrm{Var}(\hat{\mathrm{i}}^l) = \frac{1}{2}\frac{\lambda^2}{\theta^2} n_{l-1} \mathrm{Var}(\mathbf{W}^l) \mathrm{Var}(\hat{\mathrm{i}}^{l-1}) \tag{9}$$

As suggested by He et al. (2015), the reduction or magnification of forward signals in network is not ideal. So we determine the distribution of $\mathbf{W}^l$ by (for $l = 2, 3, \cdots, L$):

$$\frac{\mathrm{Var}(\hat{\mathrm{i}}^l)}{\mathrm{Var}(\hat{\mathrm{i}}^{l-1})} = \frac{1}{2}\frac{\lambda^2}{\theta^2} n_{l-1} \mathrm{Var}(\mathbf{W}^l) = 1 \tag{10}$$

So $\boldsymbol{W}^l$ here is initialized from a Gaussian distribution $\mathcal{N}(0, \frac{2\theta^2}{\lambda^2 n_{l-1}})$, where n_{l-1} is the input neuron number of layer l. As other initializers do, we apply the initialization to the first layer for simplicity.

Compared with previous work on ANNs, there is no need for neurons to accumulate the input. Thus, biases are often set to zero in ANN. For SNNs, as the response asymptote indicates a non-zero non-differential point, it is natural for biases in SNN to be initialized to that non-differential value.

4 Related Work

From the perspective of loss landscape, a good weight initialization is often located in a low loss value region (Li et al., 2018). However, training deep neural networks is never an easy thing. After

Table 2: Network performance comparison on the MNIST and CIFAR10 datasets with different initialization methods. The hyphen in the table means invalid training under a given setting.

MNIST ($k = 0.8,\ \lambda = 0.2,\ \theta = 1.0$)				
	Constant coding	Poisson coding	Retina-like coding	Event coding (N-MNIST)
Glorot's	99.18	99.30	99.04	99.36
He's	99.18	99.26	99.14	99.35
Wu's	99.18	99.28	**99.24**	99.29
Lee's	99.18	99.27	99.14	99.33
Ours	**99.27**	**99.35**	99.20	**99.39**
CIFAR10 ($k = 0.8,\ \lambda = 0.2,\ \theta = 1.0$)				
	Constant coding	Poisson coding	Retina-like coding	Event coding (CIFAR10-DVS)
Glorot's	81.59	76.69	73.75	-
He's	82.95	81.86	76.36	66.9
Wu's	83.90	82.23	78.84	67.7
Lee's	83.05	81.03	73.07	-
Ours	**87.62**	**83.06**	**82.98**	**68.2**

trails of layer-wise training of deep nets (Hinton, 2007; Bengio et al., 2006), Glorot & Bengio (2010) provided a theoretical view of preventing exploding and vanishing gradients. They estimated the standard deviation of weight from layer neuron numbers and initialized bias to zero (also known as Glorot Initialization). When Glorot initialization was proposed, traditional functions, like $sigmoid$ and $tanh$, were still commonly used activation functions. The prevalence of ReLU helped improve models (Zeiler & Fergus, 2014; Szegedy et al., 2015) but also broke the hypothesis. So He et al. (2015) extended the hypothesis to piece-wise linear. Both of the methods have a linear hypothesis of activation and are widely used in deep ANNs. Independently, Saxe et al. (2014) examined the dynamics of learning and derived random orthogonal initial conditions for deep nonlinear networks.

Considering deep SNNs, except for the depth, SNNs resemble RNNs in the meantime. The difference lies in the maintenance of hidden states. Hidden states in RNNs are associated with weights. Le et al. (2015) used identity matrix to initialize RNN weight matrix and thus maintain gradients. While SNN associates membrane potentials of adjacent timesteps by a decay scalar. For SNN, a naive idea is to scale weights to make SNN at least as good as RNN.

Recent SNN researches have also proposed some initialization methods. One can regard ANN-SNN conversion as an initialization method (Rathi et al., 2020). Note that ANN-SNN conversion involves weight scaling to get the appropriate firing rate. Besides, Lee et al. (2016) set the weights and thresholds with a manually tuned hyper-parameter. They also proposed a complex backprop error normalization. Wu et al. (2018) suggested to normalize weights sampling from a standard uniform distribution with no explanation. Lee et al. (2020b) followed what He et al. (2015) had recommended with manually constant κ.

5 Experimental Validation

5.1 Experimental setting and datasets

Here to verify the accelerating training performance and final model performance against various initialization methods, we conduct experiments on classification tasks including different codings for the MNIST dataset and CIFAR10 dataset. The MNIST dataset consists of a training set with 60000 labeled hand-written digits and a testing set of other 10000 labeled digits (Lecun et al., 1998). CIFAR10 dataset consists of 60000 32x32 color images in 10 classes, with 50000 training images and 10000 test images (Krizhevsky et al., 2009). To be fed into the SNN, the static images should be converted into spike trains. We implement constant coding, Poisson coding, retina-like coding, and event coding for both datasets. Poisson coding and constant coding are popular rate coding schemes for ANN-to-SNN conversions (Rueckauer et al., 2017; Sengupta et al., 2019). Retina-like coding mimics the integration process of a bipolar cell (Zhu et al., 2019). The N-MNIST and

CIFAR10-DVS datasets are the event coding version of MNIST and CIFAR10. N-MNIST dataset converts MNIST to spike events of size 34×34 using saccades, as presented in Orchard et al. (2015). CIFAR10-DVS dataset are comprised of 10,000 128×128 event streams converted from 32×32 frame-based images in CIFAR10 using a dynamic vision sensor (DVS) (Li et al., 2017).

Previous work has revealed that spiking network accuracy is sensitive to the choice of hyper-parameters like neuron parameters (Fang et al., 2020). So we fix the neuron parameter for a given dataset as shown in Table 2 and compare our methods with four other prevalent choices (Glorot & Bengio (2010), He et al. (2015), Lee et al. (2016) and Wu et al. (2018)). The analysis of these sensitive hyper-parameters is further presented in Sec. 6. For MNIST, we train a 5-layer spiking neural network consisting of two convolutional layers, two pooling layers, and three fully-connected layers. For the CIFAR10 dataset, we train a VGG-9 model consisting of seven convolutional layers, three pooling layers, and two fully connected layers. For the CIFAR10-DVS dataset, we add one pooling layer and a fully connected layer into the VGG-9 model due to the larger size of the data. In addition, we set the time step for the forwarding process to 20 constantly. All the models are optimized using Adam optimizer with betas set to (0.9, 0.99). Detailed training hyper-parameters are shown in the appendix.

5.2 RESULTS

Fig. 3 shows training loss and validating accuracy curves using aforementioned four datasets. To be specific, loss and accuracy curves of the initial training phase are displayed to shows the comparison with other initializations on accelerating SNN training.

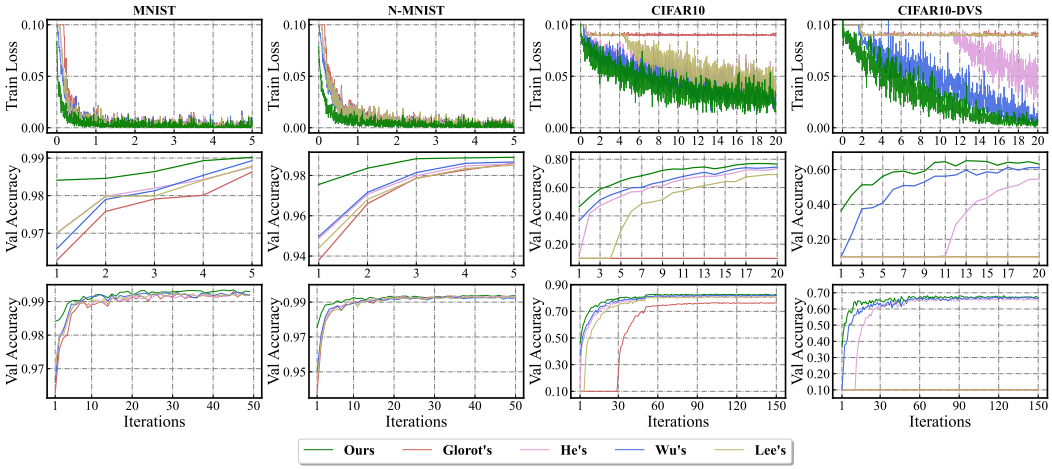


Figure 3: Training Loss and Validating accuracy curves on MNIST, NMNIST, CIFAR10, and CIFAR10-DVS datasets. MNIST and CIFAR10 datasets are encoded into Poisson distribution. First two lines shows loss and accuracy curves of the early 5 and 20 epochs on MNIST and CIFAR10 datasets. The third line shows validating accuracy curves during whole training process.

In general, Fig. 3 and Table 2 delivers that not only can good parameter initialization accelerate training but also bring some performance improvement. For MNIST, different initialization methods achieve similar performance on validating accuracy with four encoding schemes listed in Table 2, and our method performs slightly better on three of them. For CIFAR10, our method achieves a performance improvement, getting 4.67%, 1.20%, 6.62%, and 0.5% accuracy increase compared with He's method. From the first line of Fig. 3, our method enables the model to generate early gradients in fewer iterations and make the loss drop faster. In more detail, our method makes the model getting a proper parameter distribution which helps every layer coordinate with input and output that better adapts to SNN training. This advantage enables the network to begin training quickly in a few iterations. In comparison, other methods are slower than ours, for example on CIFAR10-DVS dataset method proposed in He et al. (2015) and Wu et al. (2018) drive the drop of loss effectively after about the second and the eleventh epoch.

For MNIST, our method reaches 98% validating accuracy after only 1 epoch, while Wu's, He's, and Lee's methods reach after about 2 epochs training and Glorot's reaches after 4 epochs. For N-MNIST, our method reaches 98% validating accuracy after 2 epochs, while others reach after about 3 epochs. For CIFAR10, our method first achieves 60% accuracy at about the 4th epoch, Wu's, He's, Lee's, and Glorot's reach later at about the 6th, 8th, 12th and 40th epoch successively. For CIFAR10-DVS, ours, Wu's and He's 60% accuracy at about 9th, 13th, and 25th epoch in order while Lee's and Glorot's fail training.

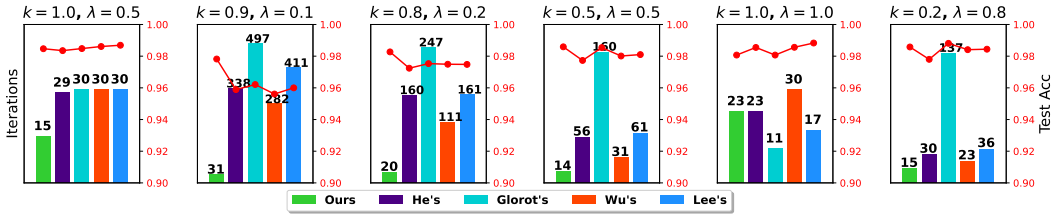
Due to the simplicity of MNIST, networks with input encoded by four schemes perform similarly. For CIFAR10, the model trained with constant coding achieves the highest accuracy for this data form is almost the same as the original data. Poisson coding performs worse than constant coding for its more complicated coding rules. Retina-like coding brings much degradation probably due to the loss of some texture information. The results show this coding scheme influences the other four initialization methods more than ours. Event data (CIFAR10-DVS) is sparse as well complicated and the amount of data is much smaller than CIFAR10, which increases the training difficulty leading to performance degradation.

6 Analysis of Empirical Results

To study whether the initialization strategy is sensitive to the hyper-parameters of the model and training, we conduct lots of experiments. The experiments use the MNIST dataset and the same 5-layer network as in Sec. 5. We first test the performance of the neuron configurations (k, λ, and θ). And then representative configurations are selected to further testing versatility over optimizers and surrogate functions. The training takes 3 epochs with a mini-batch size of 64.

6.1 Performance Analysis on Different Neurons

We perform analysis on different neuron types, that is, different neuron parameter setting. In spite that θ is fixed to 1.0, k is sampled from 0.1 to 1.0 and λ is sampled from 0.1 to 4.0. Fig. A7 compares the training speed in the grid experiments. For most (k, λ) settings, our method obtains an obvious acceleration against other methods. In particular, our method can train well for networks that cannot start up training by other initialization methods.



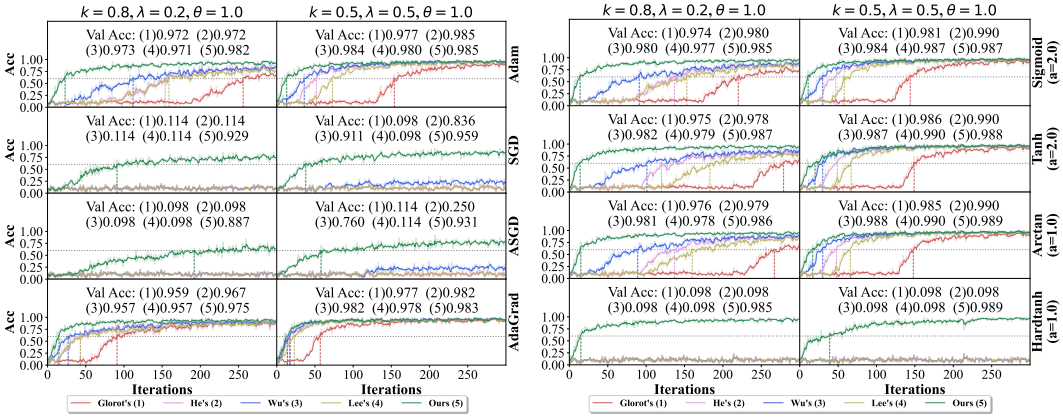
(a) Non-leaky (b) $\tau = 10$ (c) $\tau = 5$ (d) $\tau = 2$ (e) Non-leaky (f) Fast leaky

Figure 4: Validating accuracy (red polyline) and training iterations (colored histogram, the value indicates the number of iteration when accuracy first reaching the expected level 60%) for first-order integrate-and-fire neurons with different (k, λ) settings and initialization methods. Note that θ is set to 1.0.

We choose six representative groups to better illustrate this as shown in Fig. 4. For leaky neurons with different τ, with the decrease of τ, the training speed and accuracy of all initialization methods have been improved. In this case, our method maintains high accuracy and a training acceleration of more than $11\times$ compared to the baseline Glorot initialization. For non-leaky neurons of (a) and (e), the difference among methods is not obvious. As the λ decreases, our method enables faster training. For neurons that fast leak ($k = 0.2$), Glorot's performs badly on start-up training speed yet gets comparable accuracy with our method. Among the faster methods, our method achieves the best accuracy.

Besides, instead of fixing θ, we plot the heat map in Fig. A8 after sampling θ from $\{0.5, 1, 2, 4\}$. The figure illustrates that our method can fit almost every chosen parameter and help address the stall of training caused by extremely large bias due to the compensation of initial biases.

6.2 PERFORMANCE ANALYSIS ON DIFFERENT OPTIMIZERS AND SURROGATE FUNCTIONS



(a) Performance of optimizers (b) Performance of surrogate functions

Figure 5: Training accuracy curve and validating accuracy using different training hyper-parameters, different (k, λ, θ) settings and initialization methods. Performance of four different optimizers (Adam, SGD, ASGD, AdaGrad) are shown in (a). Performance of four different surrogate functions (Sigmoid, Tanh, Arctan, Hardtanh) are presented in (b).

We select $\tau = 5$ and 2 to test the sensitivity of training hyper-parameters. Two main factors that influence the gradient update are optimizer and surrogate function. Thus, Fig. 5 shows the training curves of different initialization methods, and the accuracy is marked on the figure in the order of the legends. Generally speaking, if the gradient update w.r.t. the hyper-parameter can start learning, the difference in accuracy is not huge. Although we can still observe the advantage of our method in accuracy. For example, for $(0.8, 0.2, 1.0)$, the accuracy of Wu's and Lee's methods is 1.8% lower than ours. In addition, roughly, the initialization method can observe an order of training speed from the experiment, that is, ours $>$ Wu's $>$ He's $\geq$ Lee's $>$ Glorot's. Besides, the experiments with smaller τ train better.

For optimizers, SGD(lr=0.5) and ASGD(lr=2.0) fail to activate fast early training for other initialization, yet our methods still works. Compared with Adam(lr=1e-3,β =(0.9,0.99)), AdaGrad(lr=1e-2) seems less sensitive to the choice of initialization methods.

As for surrogate functions, the order of methods is maintained. HardTanh gives coarse surrogate gradients and suppresses the training of four other methods. Surrogate functions that produce more fine-grained gradients all activate the training. Using Sigmoid(a=2.0) has some advantages in training speed, but it will be slightly inferior to Tanh(a=2.0) and Arctanh(a=1.0) in training accuracy.

7 CONCLUSIONS AND DISCUSSIONS

In this work, we aim to enhance the efficient BPTT training of SNNs by proposing an initialization method to match the response of spiking neurons in initial training. Our method bridges the spiking neuron response to the wisdom of traditional deep learning training, which may have an influence on future research like ANN-to-SNN conversions with LIF neurons or other SNN training methods. Experimental results validate that our initialization shows superiority, versatility, and robustness in the vast parameter space.

The large number of experiments we conducted allows us to provide suggestions for spiking neural network training. A good combination of hyper-parameters is essential. A good pair of neuron parameters itself can bring about high-speed and high-precision training due to better gradients. Our initialization can speed up training further and adapt to more difficult neural parameters. We recommend the use of adaptive optimization as it helps accumulate better gradients. The ArcTan and Tanh surrogate functions are suggested due to high accuracy and comparably high training speed.

REFERENCES

Larry F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pp. 153–160. MIT Press, 2006.

Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham N. Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios D. Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1): 82–99, 2018.

Michael V. DeBole, Brian Taba, Arnon Amir, Filipp Akopyan, Alexander Andreopoulos, William P. Risk, Jeff Kusnitz, Carlos Ortega Otero, Tapan K. Nayak, Rathinakumar Appuswamy, Peter J. Carlson, Andrew S. Cassidy, Pallab Datta, Steven K. Esser, Guillaume Garreau, Kevin L. Holland, Scott Lekuch, Michael Mastro, Jeffrey L. McKinstry, Carmelo di Nolfo, Brent Paulovicks, Jun Sawada, Kai Schleupen, Benjamin Shaw, Jennifer L. Klamo, Myron D. Flickner, John V. Arthur, and Dharmendra S. Modha. TrueNorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5):20–29, 2019.

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pp. 1–8. IEEE, 2015.

Jianhao Ding, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Optimal ANN-SNN conversion for fast and accurate inference in deep spiking neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 2328–2336, 2021.

Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *CoRR*, abs/2007.05785, 2020.

Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pp. 249–256, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1026–1034. IEEE Computer Society, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016.

Weihua He, Yujie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. Comparing SNNs and RNNs on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, 132:108–120, 2020.

Geoffrey Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11(10): 428–434, 2007.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Nguyen-Dong Ho and Ik-Joon Chang. TCL: an ANN-to-SNN conversion with trainable clipping layers. *CoRR*, abs/2008.04509, 2020.

Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

Ionel-Alexandru Hosu and Traian Rebedea. Playing Atari games with deep reinforcement learning and human checkpoint replay. *CoRR*, abs/1607.05077, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4107–4115, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114, 2012.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Chankyu Lee, Adarsh Kosta, Alex Zihao Zhu, Kenneth Chaney, Kostas Daniilidis, and Kaushik Roy. Spike-flownet: Event-based optical flow estimation with energy-efficient hybrid neural networks. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIX*, volume 12374 of *Lecture Notes in Computer Science*, pp. 366–382. Springer, 2020a.

Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020b.

Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 6391–6401, 2018.

Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. CIFAR10-DVS: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017.

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

Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4):593–616, 2004.

Julien Mairal. Optimization with first-order surrogate functions. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 783–791. JMLR.org, 2013.

Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9:437, 2015.

Eustace Painkras, Luis A. Plana, Jim D. Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber. SpiNNaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, 2013.

Priyadarshini Panda, Sai Aparna Aketi, and Kaushik Roy. Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization. *Frontiers in Neuroscience*, 14, 2020.

Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:682, 2017.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: VGG and residual architectures. *Frontiers in Neuroscience*, 13:95, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018.

Zheyu Yang, Yujie Wu, Guanrui Wang, Yukuan Yang, Guoqi Li, Lei Deng, Jun Zhu, and Luping Shi. DashNet: A hybrid artificial and spiking neural network for high-speed object tracking. *CoRR*, abs/1909.12942, 2019.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, volume 8689 of *Lecture Notes in Computer Science*, pp. 818–833. Springer, 2014.

Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 11062–11070. AAAI Press, 2021.

Lin Zhu, Siwei Dong, Tiejun Huang, and Yonghong Tian. A retina-inspired sampling method for visual texture reconstruction. In *IEEE International Conference on Multimedia and Expo, ICME 2019, Shanghai, China, July 8-12, 2019*, pp. 1432–1437. IEEE, 2019.

A APPENDIX

A.1 APPENDIX FOR THEORETICAL ANALYSIS

Here we provide the theoretic proofs of theorems in Sec. 2. These are the foundation of the analyses in this paper. Also, Fig. A6 is presented here to illustrate the surrogate function and symmetric gradient.

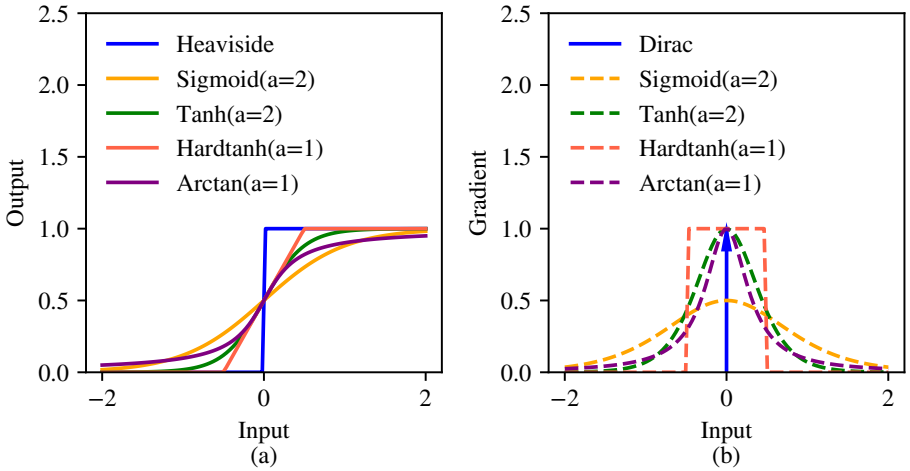


Figure A6: Surrogate functions and corresponding gradients.

Table A3: Surrogate functions for non-differential model (a is the parameter).

Surrogate function h	$h(x, a)$	$h'(x, a)$
Sigmoid-like	$1/(1 + e^{-ax})$	$a \cdot h(x, a)(1 - h(x, a))$
Tanh-like	$\frac{1}{2}\tanh(ax) + \frac{1}{2}$	$\frac{1}{2}a \cdot (1 - \tanh(ax)^2)$
Hardtanh-like	$\mathrm{clip}(ax + \frac{1}{2}, 0, 1)$	$a \cdot \mathrm{Heaviside}(\frac{1}{2a} - \lvert x \rvert)$
Arctan-like	$\frac{1}{\pi}\arctan(\pi a x) + \frac{1}{2}$	$a/(1 + (\pi a x)^2)$

Theorem 1. *For first-order integrate-and-fire neurons expressed in Eq. 2 ($u^t = kv^t + \lambda i^t, k \in (0, 1]$), the model takes constant current $i^t = i$ as input and output spikes. Denote the discrete output firing rate w.r.t. i as $f = \frac{T}{N}$, where T is the total inference time and N is the spike count. Then the input-output response curve (f-i curve) is estimated by:*

$$
\begin{aligned}
&\textit{If } k = 1: \\
&\quad f = \begin{cases} 0, & i \leq 0, \\ \frac{\lambda}{\theta} i, & 0 < i < \frac{\theta}{\lambda}, \\ 1, & i \geq \frac{\theta}{\lambda}. \end{cases} \\
&\textit{If } k \in (0, 1): \\
&\quad f = \begin{cases} 0, & i \leq 0, \\ (k-1)/\ln(1 + \frac{(k-1)\theta}{\lambda i}), & 0 < i < \frac{\theta}{\lambda} \cdot \frac{k-1}{e^{k-1}-1}, \\ 1, & i \geq \frac{\theta}{\lambda} \cdot \frac{k-1}{e^{k-1}-1}. \end{cases}
\end{aligned}
\tag{A1}
$$

Proof. For Eq. 2, the iterative functions are first converted to differential equation. Here we use the scalar variable form because we are interested in the interval time between resting state and firing state for constant input. Our goal is to work out the time t^* of firing ($u(t^*) = \theta$) with $u(0) = 0$.

And, the discrete output firing rate $f = \frac{T}{N}$ can be approximated by the inverse of spiking interval t^*.

$$\frac{du}{dt} = (k-1)u + \lambda i \tag{A2}$$

One can use integral to get the expression of u when given constant input. When $k = 1$, the integral is only an accumulation of the scaled input. Considering $u(0) = 0$, we have:

$$u(t) = \lambda i \cdot t. \tag{A3}$$

Then, with $u(t^*) = \theta$, it is calculated that $t^* = \frac{\theta}{\lambda i}$. Thus, when $k = 1$,

$$f = \frac{T}{N} \approx \frac{1}{t^*} = \frac{\lambda}{\theta} i. \tag{A4}$$

For $i \geq \frac{\theta}{\lambda}$, neurons constantly fire, and f reaches its upper bound 1. When $i \leq 0$, neurons accumulate negative input and produce zero spike, $f = 0$. When $k \in (0, 1)$, Eq. A2 is an ordinary differential equation. By separating integral variables u and t, we have:

$$\frac{du}{(k-1)u + \lambda i} = dt.$$

Integrate two sides of the equation, and we obtain

$$\ln((1-k)u - \lambda i) = (k-1)t + C$$
$$(1-k)u - \lambda i = C' e^{(k-1)t},$$

where C and C' are the constant of integration. Considering $u(0) = 0$, we have $C' = -\lambda i$. And thus,

$$u(t) = \frac{\lambda i}{1-k}(1 - e^{-(1-k)t}). \tag{A5}$$

Then, by solving $u(t^*) = \theta$, $t^* = \frac{1}{k-1} \cdot \ln(1 + \frac{(k-1)\theta}{\lambda i})$. Hence,

$$f = \frac{T}{N} \approx \frac{1}{t^*} = (k-1)/\ln(1 + \frac{(k-1)\theta}{\lambda i}). \tag{A6}$$

When $i = \frac{\theta}{\lambda} \frac{k-1}{e^{k-1}-1}$, $f = 1$. Thus, when $i > \frac{\theta}{\lambda} \frac{k-1}{e^{k-1}-1}$, f is clipped to 1. □

Theorem 2. *For first-order integrate-and-fire neurons expressed in Eq. 2 ($u^t = kv^t + \lambda i^t, k \in (0, 1]$), the slant asymptote of f-i satisfying $f \in (0, 1)$ exists and can be uniformed as:*

$$f_a = \frac{\lambda}{\theta} i + \frac{k-1}{2}. \tag{A7}$$

Proof. From Theorem 2, the relationship of i and f satisfying $f \in (0, 1)$ is given by:

$$f = \begin{cases} \frac{\lambda}{\theta} i, & k = 1, \\ (k-1)/\ln(1 + \frac{(k-1)\theta}{\lambda i}), & k \in (0, 1). \end{cases} \tag{A8}$$

Assume that the asymptote is formulated as $f_a = K \cdot i + B$. Then, for $k = 1$, $K = \frac{\lambda}{\theta}$, and $B = 0$. For $k \in (0, 1)$, if a slant asymptote exists, then $\lim_{i \to \infty} \frac{f}{i} \to K$, and $\lim_{i \to \infty} [f - Ki] \to B$.

$$\begin{aligned}
\lim_{i \to \infty} \frac{f}{i} &= \lim_{i \to \infty} \frac{k-1}{i \ln(1 + \frac{\theta(k-1)}{\lambda i})} \\
&= \lim_{i \to \infty} \frac{\frac{k-1}{i}}{\ln(\lambda i + \theta(k-1)) - \ln(\lambda i)} \\
&= \lim_{i \to \infty} \frac{-\frac{k-1}{i^2}}{\frac{\lambda}{\lambda i + \theta(k-1)} - \frac{\lambda}{\lambda i}} \quad \text{(L'Hospital's Rule)} \\
&= \lim_{i \to \infty} \frac{\lambda i + \theta(k-1)}{\theta i} \\
&= \frac{\lambda}{\theta} \quad \text{(L'Hospital's Rule)}
\end{aligned}$$

Thus a slant asymptote exists for $k \in (0,1)$, and $K = \frac{\lambda}{\theta}$ for all $k \in (0,1]$. For $k \in (0,1)$,

$$
\begin{aligned}
B &= \lim_{i\to\infty}\left[\frac{k-1}{\ln(1+\frac{\theta(k-1)}{\lambda i})} - \frac{\lambda}{\theta}i\right] \\
&= \lim_{i\to\infty}\frac{k-1-\frac{\lambda}{\theta}i[\ln(\lambda i+\theta(k-1))-\ln(\lambda i)]}{\ln(\lambda i+\theta(k-1))-\ln(\lambda i)} \\
&= \lim_{i\to\infty}\frac{-\frac{\lambda}{\theta}\ln(\lambda i+\theta(k-1)) - \frac{\lambda}{\theta}\frac{\lambda i}{\lambda i+\theta(k-1)} + \frac{\lambda}{\theta}\ln(\lambda i) + \frac{\lambda}{\theta}}{\frac{\lambda}{\lambda i+\theta(k-1)} - \frac{\lambda}{\lambda i}} \quad \text{(L'Hospital's Rule)} \\
&= \lim_{i\to\infty}\frac{\lambda}{\theta}\frac{\ln(\lambda i)-\ln(\lambda i+\theta(k-1))+1-\frac{\lambda i}{\lambda i+\theta(k-1)}}{\frac{\lambda}{\lambda i+\theta(k-1)} - \frac{\lambda}{\lambda i}} \\
&= \lim_{i\to\infty}\frac{\lambda}{\theta}\frac{\frac{1}{i} - \frac{\lambda(\lambda i+\theta(k-1))+\lambda\theta(k-1)}{(\lambda i+\theta(k-1))^2}}{-\frac{\lambda^2}{(\lambda i+\theta(k-1))^2}+\frac{1}{i^2}} \quad \text{(L'Hospital's Rule)} \\
&= \lim_{i\to\infty}\frac{\lambda}{\theta}\frac{\theta^2(k-1)^2 i}{2\lambda\theta(k-1)i+\theta^2(k-1)^2} \\
&= \frac{\lambda}{\theta}\frac{\theta(k-1)}{2\lambda} \quad \text{(L'Hospital's Rule)} \\
&= \frac{k-1}{2}
\end{aligned}
$$

When $k = 1$, $B = 0$. Thus, $f_a = \frac{\lambda}{\theta}i + \frac{k-1}{2}$ is the uniformed slant asymptote for all $k \in (0,1]$.

□

A.2 Appendix for experimental results

The detailed hyper-parameters are presented in Table A4. Note that the data in CIFAR10-DVS and N-MNIST take the form of address event representation (AER). To split the events into T slices that match the input steams of our network, we utilize the technique referenced in Appendix F, Fang et al. (2020) to pre-process the data.

Performance Analysis on Different Neurons

As Eq. 2 presents, neuron types and firing patterns can be controlled by k and λ which may vary in the biological brain. To cover neuron models used in existing neuroscience literature as much as possible, we sample k from 0.1 to 1.0 and λ from 0.1 to 4.0 discretely. In each experiment, we compare four methods with our initialization method from two aspects: training speed and accuracy. The four methods are respectively proposed in Glorot & Bengio (2010), He et al. (2015), Lee et al. (2016) and Wu et al. (2018).

The exact iteration step is recorded when training accuracy first reaches an expected level (60%), which can be representative of training speed. As shown in Fig. A7, green bars represent our method, while bars in other colors represent others. It is intuitive that our method reaches the expected accuracy faster or at least in the same performance level for most neuron types compared to other methods, which is more obvious when λ falls in the range of $(-\infty, 0.6)$ and $(2.0, +\infty)$. Especially, there are even no valid records using other methods in some cases that λ is set to 0.1, 0.2, 8.0, and so on. While in these cases the training accuracy of our method still reaches the expected level in a very short time.

Apart from the superiority of training speed, we also measure performance on final testing accuracy to check robustness on classification tasks in various configurations. As shown in Fig. A8, classification accuracies of five initialization methods with different configurations of neurons are shown in the heat map. Our method can effectively and efficiently train SNN in almost all neurons types while other methods fail at training the network in some neuron types.

Table A4: Detailed hyper-parameter setting in experimental validation. The network is designed to fit the different shape of data input.

	MNIST	N-MNIST	CIFAR10	CIFAR10-DVS
Input size	28×28	T$\times 32 \times 32$	32×32	T$\times 128 \times 128$
Timestep	20	20	20	20
Network	conv $3 \times 3 \times 32$ LIF neuron pool 2×2 conv $3 \times 3 \times 32$ LIF neuron pool 2×2 fc 500 LIF neuron fc 100 LIF neuron fc 10	conv $3 \times 3 \times 32$ LIF neuron pool 2×2 conv $3 \times 3 \times 32$ LIF neuron pool 2×2 fc 500 LIF neuron fc 100 LIF neuron fc 10	conv $3 \times 3 \times 128$ LIF neuron conv $3 \times 3 \times 128$ LIF neuron pool 2×2 conv $3 \times 3 \times 256$ LIF neuron conv $3 \times 3 \times 256$ LIF neuron pool 2×2 conv $3 \times 3 \times 256$ LIF neuron conv $3 \times 3 \times 256$ LIF neuron conv $3 \times 3 \times 256$ LIF neuron pool 2×2 fc 1024 LIF neuron Dropout fc 10	conv $3 \times 3 \times 64$ LIF neuron conv $3 \times 3 \times 128$ LIF neuron pool 2×2 conv $3 \times 3 \times 128$ LIF neuron conv $3 \times 3 \times 128$ LIF neuron pool 2×2 conv $3 \times 3 \times 128$ LIF neuron pool 2×2 conv $3 \times 3 \times 128$ LIF neuron pool 2×2 conv $3 \times 3 \times 128$ LIF neuron pool 2×2 fc 1024 LIF neuron Dropout fc 1024 LIF neuron Dropout fc 10
Epoch	50	50	150	150
Batch size	32	32	32	16
LR	0.001	0.001	0.001	0.001
Optimizer	Adam(0.9,0.99)	Adam(0.9,0.99)	Adam(0.9,0.99)	Adam(0.9,0.99)
Scheduler	Cosine(T_{max}=50)	Cosine(T_{max}=50)	MultiStep (T=[50,90,130])	MultiStep (T=[50,90,130])
Surrogate	ArcTan(a=2.0)	ArcTan(a=2.0)	ArcTan(a=2.0)	ArcTan(a=2.0)

Eight representative groups of (k, λ, θ) are selected to further analyze whether the proposed method obtain versatility on the choices of optimizers and surrogate functions, which are (0.3, 0.1, 1.0), (0.8, 0.2, 1.0), (0.8, 0.5, 1.0), (0.2, 0.8, 1.0), (1.0, 1.0, 1.0), (0.5, 4.0, 1.0), (0.6, 0.3, 4.0) and (0.4, 0.4, 4.0).

Versatility on Optimizers

During training, the model uses different optimizers to minimize the loss function. To test the versatility of our method on various optimizers, we choose four classic and prevalent optimizers to train the model, i.e. Adam, SGD, ASGD, and AdaGrad. Detailed optimizer parameters are listed in Table A5.

We present the final accuracy of training in Table A7. In Fig. A9, we show the training curve in the first 500 steps to deliver the result. As shown in the figure, when SGD or ASGD is chosen to optimize the network, our method demonstrates an advantage in accelerating the training in the start-

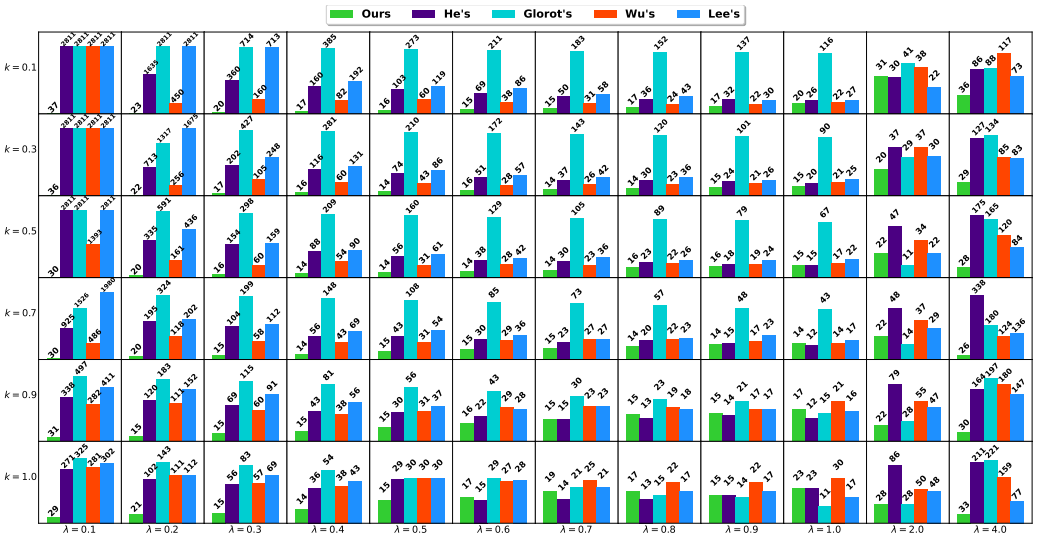


Figure A7: Training iterations when accuracy first reaching the expected level (60%) for first-order integrate-and-fire neurons with different (k, λ) settings and initialization methods. Note that θ is set to 1.0. The total number of iteration of 3 training epochs is 2811. The value 2811 in the figure means the setting cannot train the network to reach the expected training accuracy level.

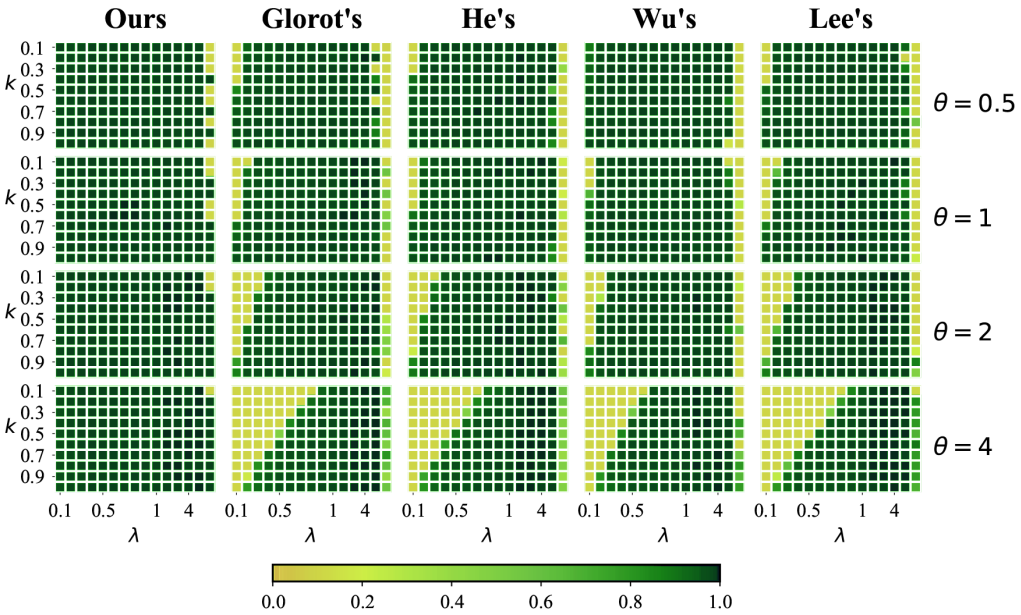


Figure A8: Training accuracy heat map for first-order integrate-and-fire neurons with different (k, λ, θ) settings and initialization methods. The color represents the final validating accuracy with given setting.

up phase with the accuracy reaching a good level within 100 iterations in most cases. For example, when $k = 0.8$, $\lambda = 0.5$, and $\theta = 1.0$, Wu's and He's methods begin training early but with slower speed, achieving around 0.9 accuracy after 500 iterations, while Glorot's and Lee's methods show much worse performance. When Adam or AdaGrad is chosen to optimize the network, despite the similar performance of all methods in some cases (for example 3-5th rows in the figure), our method outperforms others on accelerating the training. Besides, our weight initialization for SNN enables valid training in some hard cases when others totally fail in training like $(k = 0.4, \lambda = 0.4, \theta = 0.4)$.

Table A5: Optimizers and their parameters used to test the versatility.

Optimizer	Learning rate	Other Parameters
Adam	0.001	betas=(0.9, 0.99)
SGD	0.5	momentum=0.9
ASGD	2.0	lambda=0.000001, alpha=0.75
AdaGrad	0.01	-

Versatility on Surrogate Functions

Various surrogate functions are chosen in SNN training w.r.t. different needs and tasks. As listed in Table A3, the four surrogate functions (Sigmoid, Tanh, Hardtanh, Arctan) are popular using back-propagation through time. As described in Sec. 2.1, the input uses the Heaviside function to perform forward pass, while the backward gradient is computed using surrogate functions. We train the 5-layer spike neuron network using these surrogate functions consistent with that in Fig. A3 with adjusting parameter a.

The final validating accuracy of 3 epoch training is presented in Table A7. And the training curve of the first 220 steps of four surrogate functions is shown in Fig. A10. The curve in green shows our method's effectiveness. Our method reach acceptable accuracy within short iteration steps (less than 50 in most cases) While in comparison other methods start the decline of network loss slower, taking parameters of (0.8, 0.2, 1.0), (0.8, 0.5, 1.0), (0.2, 0.8, 1.0) and (0.5, 4.0, 1.0) as examples. Especially when k and λ are small values or θ is a large value, other initialization methods cannot generate proper gradients for back-propagating leading to the training failure. For almost all sets of surrogate functions and neuronal parameters, our method enables neurons firing and the network generating gradient early. Fig. A10 reveal that our method enables models to start training faster ending with considerably great classification accuracy.

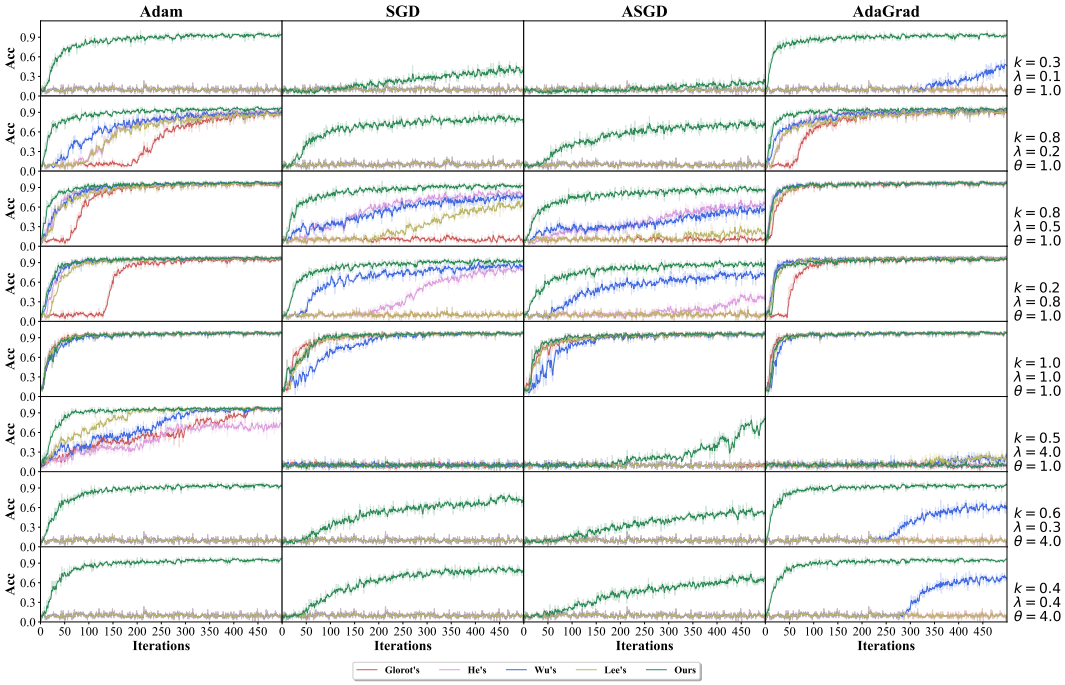


Figure A9: Training accuracy curve using four different optimizers, different (k, λ, θ) settings and initialization methods.

Table A6: Validating accuracy training MNIST with 3 epochs using four different optimizers, different (k, λ, θ) settings and initialization methods. The hyphen in the table means invalid training under given setting.

	$k = 0.3,\ \lambda = 0.1,\ \theta = 1.0$					$k = 0.8,\ \lambda = 0.2,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Adam	0.979	-	-	-	-	0.982	0.972	0.972	0.973	0.971
SGD	0.776	-	-	-	-	0.929	-	-	-	-
ASGD	0.636	-	-	-	-	0.887	-	-	-	-
AdaGrad	0.966	0.878	0.898	0.876	0.740	0.975	0.959	0.967	0.956	0.957
	$k = 0.8,\ \lambda = 0.5,\ \theta = 1.0$					$k = 0.2,\ \lambda = 0.8,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Adam	0.984	0.980	0.986	0.983	0.984	0.984	0.978	0.988	0.982	0.989
SGD	0.975	0.921	0.959	0.964	0.945	0.968	-	0.950	0.956	0.884
ASGD	0.958	0.165	0.910	0.918	0.866	0.951	-	0.905	0.926	0.612
AdaGrad	0.984	0.978	0.984	0.984	0.982	0.977	0.978	0.986	0.985	0.986
	$k = 1.0,\ \lambda = 1.0,\ \theta = 1.0$					$k = 0.5,\ \lambda = 4.0,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Adam	0.987	0.986	0.987	0.986	0.986	0.986	0.987	0.984	0.985	0.986
SGD	0.985	0.982	0.985	0.983	0.985	-	-	-	-	-
ASGD	0.986	0.981	0.986	0.981	0.984	0.981	-	-	-	-
AdaGrad	0.986	0.984	0.986	0.985	0.986	0.983	-	-	0.977	0.984
	$k = 0.6,\ \lambda = 0.3,\ \theta = 4.0$					$k = 0.4,\ \lambda = 0.4,\ \theta = 4.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Adam	0.980	-	-	-	-	0.984	-	-	-	-
SGD	0.895	-	-	-	-	0.923	-	-	-	-
ASGD	0.841	-	-	-	-	0.876	-	-	-	-
AdaGrad	0.968	0.882	0.889	0.910	0.743	0.972	0.906	0.924	0.926	0.752

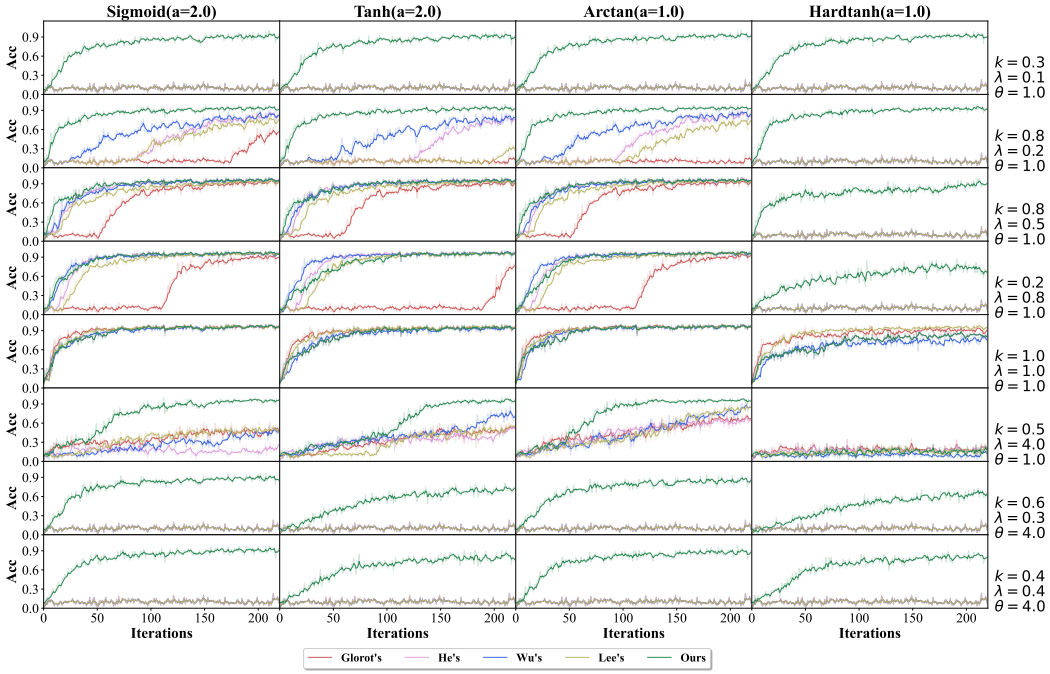


Figure A10: Training curve using four different surrogate functions, different (k, λ, θ) settings and initialization methods. The dash line in the table means invalid training under given setting.

Table A7: Validating accuracy training MNIST with 3 epochs using four different surrogate functions, different (k, λ, θ) settings and initialization methods. The hyphen in the table means invalid training under given setting.

	$k = 0.3,\ \lambda = 0.1,\ \theta = 1.0$					$k = 0.8,\ \lambda = 0.2,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Hardtanh(a=1.0)	0.982	-	-	0.310	-	0.985	0.974	0.980	0.980	0.977
Arctan(a=1.0)	0.983	-	-	-	-	0.984	0.972	0.980	0.980	0.976
Tanh(a=2.0)	0.984	-	-	-	-	0.986	0.976	0.979	0.981	0.978
Sigmoid(a=2.0)	0.984	-	-	-	-	0.985	-	-	-	-
	$k = 0.8,\ \lambda = 0.5,\ \theta = 1.0$					$k = 0.2,\ \lambda = 0.8,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Hardtanh(a=1.0)	0.987	0.985	0.987	0.984	0.986	0.987	0.983	0.990	0.986	0.988
Arctan(a=1.0)	0.988	0.983	0.986	0.987	0.989	0.988	0.986	0.991	0.987	0.991
Tanh(a=2.0)	0.989	0.987	0.987	0.988	0.990	0.988	0.984	0.990	0.989	0.991
Sigmoid(a=2.0)	0.988	-	-	-	-	0.986	-	-	-	-
	$k = 1.0,\ \lambda = 1.0,\ \theta = 1.0$					$k = 0.5,\ \lambda = 4.0,\ \theta = 1.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Hardtanh(a=1.0)	0.988	0.988	0.988	0.984	0.989	0.985	0.987	0.980	0.984	0.986
Arctan(a=1.0)	0.986	0.987	0.986	0.981	0.984	0.987	0.987	0.973	0.973	0.978
Tanh(a=2.0)	0.988	0.988	0.988	0.990	0.987	0.988	0.983	0.985	0.984	0.980
Sigmoid(a=2.0)	0.987	0.986	0.987	0.887	0.989	0.199	0.197	-	-	0.176
	$k = 0.6,\ \lambda = 0.3,\ \theta = 4.0$					$k = 0.4,\ \lambda = 0.4,\ \theta = 4.0$				
	Ours	Glorot's	He's	Wu's	Lee's	**Ours**	Glorot's	He's	Wu's	Lee's
Hardtanh(a=1.0)	0.979	-	-	-	-	0.983	-	-	-	-
Arctan(a=1.0)	0.917	-	-	-	-	0.953	-	-	-	-
Tanh(a=2.0)	0.968	-	-	-	-	0.976	-	-	-	-
Sigmoid(a=2.0)	0.887	-	-	-	-	0.951	-	-	-	-