# Membrane Potential Batch Normalization for Spiking Neural Networks

Yufei Guo[*], Yuhan Zhang[*], Yuanpei Chen, Weihang Peng, Xiaode Liu, Liwen Zhang,
Xuhui Huang, Zhe Ma[†]
Intelligent Science & Technology Academy of CASIC, China
Scientific Research Laboratory of Aerospace Intelligent Systems and Technology, China

## Abstract

*As one of the energy-efficient alternatives of conventional neural networks (CNNs), spiking neural networks (SNNs) have gained more and more interest recently. To train the deep models, some effective batch normalization (BN) techniques are proposed in SNNs. All these BNs are suggested to be used after the convolution layer as usually doing in CNNs. However, the spiking neuron is much more complex with the spatio-temporal dynamics. The regulated data flow after the BN layer will be disturbed again by the membrane potential updating operation before the firing function, i.e., the nonlinear activation. Therefore, we advocate adding another BN layer before the firing function to normalize the membrane potential again, called MPBN. To eliminate the induced time cost of MPBN, we also propose a training-inference-decoupled re-parameterization technique to fold the trained MPBN into the firing threshold. With the re-parameterization technique, the MPBN will not introduce any extra time burden in the inference. Furthermore, the MPBN can also adopt the element-wised form, while these BNs after the convolution layer can only use the channel-wised form. Experimental results show that the proposed MPBN performs well on both popular non-spiking static and neuromorphic datasets.*

## 1. Introduction

Emerged as a biology-inspired method, spiking neural networks (SNNs) have received much attention in artificial intelligence and neuroscience recently [17, 13, 57, 56, 47, 58, 59]. SNNs deal with binary event-driven spikes as their activations and therefore the multiplications of activations and weights can be substituted for additions or only keeping silents. Benefitting from such a computation paradigm, SNNs derive extreme energy efficiency and run efficiently when implemented on neuromorphic hardware [1, 42, 5].

Despite the SNN has achieved great success in diverse fields including pattern recognition [12, 21, 19, 14], object detection [30], language processing [55], robotics [9], and so on, its development is deeply inspired by the experience of convolutional neural networks (CNNs) in many aspects. However, the spiking neuron model along with the rich spatio-temporal dynamic makes SNNs much different from CNNs, and directly transferring some experience of CNNs to SNNs without any modifications may be not a good idea. As one of the famous techniques in CNNs, the batch normalization(BN) technique shows great advantages. It can reduce the gradient exploding/vanishing problem, flatten the loss landscape, and reduce the internal covariate shift, thus being widely used in CNNs. There are also some works trying to apply normalization approaches in the SNN field to help model convergence. For example, inspired by BN in CNNs, NeuNorm [51] was proposed to normalize the data along the channel dimension. Considering that the temporal dimension is also important in SNNs, threshold-dependent batch normalization (tdBN) [62] then extended the scope of BN to the additional temporal dimension. Subsequently, to better depict the differences of data flow distributions in different time dimensions, the temporal batch normalization through time (BNTT) [31], postsynaptic potential normalization (PSP-BN) [28], and temporal effective batch normalization (TEBN) [10] that regulate the data flows with multiple BNs on different time steps were proposed.

However, all these BNs proposed in SNNs are advised to be used after convolution layers as usually doing in CNNs. This ignores the fact that the nonlinear transformation in the SNN spiking neuron is much more complex than that of the ReLU neuron. In the spiking neuron, the data flow after the convolution layer will be first injected into the residual membrane potential (MP) coming from the previous time step to generate a new MP at the current time step. And then the neuron will fire a spike or keep silent still based on whether or not the new MP is up to the firing threshold. Obviously, though the data flow has been normalized by the BN after the convolution layer, it will be

[*]Equal contribution.
[†]Corresponding author, mazhe_thu@163.com.

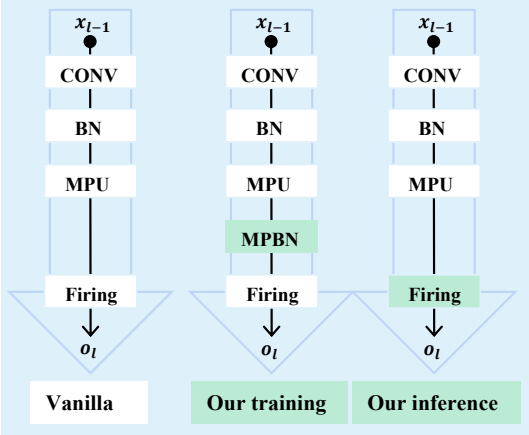



Figure 1: The difference between our SNN with MPBN and the vanilla SNN. We add another BN layer after membrane potential updating (MPU) operation in the training. The MPBN can be folded into the firing threshold and then the homogenous firing threshold will be transformed into different ones.

disturbed again by the residual MP in the membrane potential updating process. Therefore, we advocate also adding a BN layer after MP updating to regulate the data flow once again, called MPBN. Furthermore, we also propose a training-inference-decoupled re-parameterization technique in SNNs to fold the trained MPBN into the firing threshold. Hence, the MPBN will not induce any extra burden in the inference but a trivial burden in the training. The MPBN can be extended to channel-wised MPBN and element-wised MPBN further, which is very different from that of CNNs where only channel-wised normalization can be folded into weights. The difference between our SNN with MPBN and the vanilla SNN is illustrated in Fig. 1. Our main contributions are as follows:

- We propose to add another BN layer after the membrane potential updating operation named MPBN to handle the data flow disturbance in the spiking neuron. The experiment shows that MPBN can flatten the loss landscape further, thus benefiting model convergence and task accuracy.
- We also propose a re-parameterization method to decouple the training-time SNN and the inference-time SNN. In specific, we propose a method to fold the trained MPBN parameter into the firing threshold. Therefore, MPBN can be seen as only training auxiliary manner free from burdens in the inference-time. This re-parameterization method is suitable for both channel-wised MPBN and element-wised MPBN.
- Extensive experiment results show that the SNN trained with the MPBN is highly effective compared with other state-of-the-art SNN models on both static and dynamic datasets, e.g., 96.47% top-1 accuracy and 79.51% top-1 accuracy are achieved on the CIFAR-10 and CIFAR-100 with only 2 time steps.

## 2. Related Work

### 2.1. Learning of Spiking Neural Networks

There are three kinds of learning algorithms of SNNs, including unsupervised learning [43, 24], converting ANN to SNN (ANN2SNN) [46, 25, 26], and supervised learning [19, 38, 20]. Unsupervised learning adopts some biological mechanism to update the SNN model, i.e., the spike-timing-dependent plasticity (STDP) approach [39], thus being considered a biologically plausible method. However, STDP cannot help train large-scale networks yet, thus it is usually limited to small datasets and non-ideal performance. The ANN-SNN conversion approach [22, 37] obtains an SNN by reusing well-trained homogeneous ANN parameters and replacing the ReLU neuron with a spiking neuron. Since the ANN model is easier to train and reach high performance, the ANN-SNN conversion method provides an interesting way to generate an SNN in a short time with competitive performance. However, the converted SNN will lose the rich temporal dynamic behaviors and thus cannot handle neuromorphic datasets well. Supervised learning [11, 50, 18] adopts the surrogate gradient (SG) approach to train SNNs with error backpropagation. It can handle temporal data and provide decent performance with few time steps on the large-scale dataset, thus having received much attention recently. For a more detailed introduction, please refer to the recent SNN survey [17]. Our work falls under the supervised learning.

### 2.2. Normalization in Spiking Neural Networks

The batch normalization technique was originally introduced as a kind of training auxiliary method by [29] in CNNs. It uses the weight-summed input over a mini-batch of training cases to compute a mean and variance and then uses them to regulate the summed input. This simple operation can derive many benefits. i) It reduces the internal covariate shift (ICS), thus accelerating the training of a deep neural network. ii) It makes the network insensitive to the scale of the gradients, thus a higher learning rate can be chosen to accelerate the training. iii) It makes the network suitable for more nonlinearities by preventing the network from getting stuck in the saturated modes. With these advantages, more kinds of BNs were proposed, including layer normalization [2], group normalization [52], instance normalization [48], and switchable normalization [40].

There are also some works that modify and apply normalization approaches in the SNN field. For example, NeuNorm [51] also normalizes the feature map along the chan-

nel dimension like BN in CNNs. Recently, some methods were proposed to normalize the feature map from both the channel dimension and temporal dimension to take care of the spatio-temporal characteristics of the SNN, such as the threshold-dependent batch normalization (tdBN) [62]. It extends the scope of BN to the additional temporal dimension by adopting a 3DBN-like normalization method in CNNs. Note that, the tdBN can be folded into the weights, thus inducing no burden in the inference time. Nevertheless, NeuNorm and tdBN still use the shared parameters along the temporal dimension. Some works argued that the distributions of data in different time steps vary wildly and that using shared parameters is not a good choice. Subsequently, the temporal batch normalization through time (BNTT) [31], postsynaptic potential normalization (PSP-BN) [28], and temporal effective batch normalization (TEBN) [10] were proposed. These BNs regulate the data flow utilizing different parameters through time steps. Though these BNs with different BN parameters on different time steps can train more well-performed SNN models, their parameters can not be folded into the weights, thus will increase the computations and running time in the inference.

Nevertheless, all these BNs in the SNN field are advised to be used after convolution layers. However, the data flow after the convolution layer will not be presented to the firing function directly but to the membrane potential updating function first. Hence, the data flow will be disturbed again before reaching the firing function. To this end, in this paper, we add another BN after the membrane potential updating function, called the MPBN to retain normalized data flow before the firing function.

# 3. Preliminary

## 3.1. Leaky Integrate-and-Fire Model

Different from CNNs, SNNs use binary spikes to transmit information. In the paper, we use the widely used Leaky-Integrate-and-Fire (LIF) neuron model [41] to introduce the unique spatial-temporal dynamic of the spiking model. First, we introduce the notation rules used here as follows. Vectors or tensors are denoted by bold italic letters, i.e., $\boldsymbol{x}$ and $\boldsymbol{o}$ represent the input and output variables respectively. Matrices are denoted by bold capital letters. For instance, $\mathbf{W}$ is the weight matrix. The constant is denoted by small letters.

In LIF, the membrane potential is updated by

$$\boldsymbol{u}^{(t+1),\text{pre}} = \tau \boldsymbol{u}^{(t)} + \boldsymbol{c}^{(t+1)}, \text{where } \boldsymbol{c}^{(t+1)} = \mathbf{W}\boldsymbol{x}^{(t+1)}, \tag{1}$$

where $\boldsymbol{u}$ represents the membrane potential and $\boldsymbol{u}^{(t+1),\text{pre}}$ is the updated membrane potential at time step $t+1$, $\boldsymbol{c}^{(t+1)}$ is the pre-synaptic input at time step $t+1$, which is charged by weight-summed input spikes $\boldsymbol{x}^{(t+1)}$, and $\tau$ is a constant within $(0, 1)$, which controls the leakage of the membrane potential. Then, when the updated membrane potential $\boldsymbol{u}^{(t+1),\text{pre}}$ is up to the firing threshold $V_{\text{th}}$, the LIF spiking neuron will fire a spike as bellow,

$$\begin{aligned} \boldsymbol{o}^{(t+1)} &= \begin{cases} 1 & \text{if } \boldsymbol{u}^{(t+1),\text{pre}} > V_{\text{th}} \\ 0 & \text{otherwise} \end{cases}, \\ \boldsymbol{u}^{(t+1)} &= \boldsymbol{u}^{(t+1),\text{pre}} \cdot (1 - \boldsymbol{o}^{(t+1)}). \end{aligned} \tag{2}$$

After firing, the spike output $\boldsymbol{o}^{(t+1)}$ at time step $t+1$ will be transmitted to the next layer and become its input. At the same time, the updated membrane potential will be reset to zero and becomes $\boldsymbol{u}^{(t+1)}$ to join the neuron processing at the next time step.

**The Classifier in the SNN.** In a classification model, the final output is used to compute the $\mathrm{Softmax}$ and predict the desired class object. In an SNN model, if we also use LIF neurons at the output layer to fire spikes and use the number of spikes to compute the probability, too much information will be lost. Therefore, we only integrate the output and do not fire them across time, as doing in recent work [20, 21, 12].

$$\boldsymbol{o}_{\text{out}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{c}_{\text{out}}^{(t)} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{W}\boldsymbol{x}^{(t)}. \tag{3}$$

Then, the cross-entropy loss is computed based on the true label and $\mathrm{Softmax}(\boldsymbol{o}_{\text{out}})$.

## 3.2. Batch Normalization in SNNs

Batch normalization can effectively reduce the internal covariate shift and alleviate the gradient vanishing or explosion problem for training networks, thus having been widely used in CNNs. Fortunately, BN can also be used in SNNs. Considering a spiking neuron with input $\boldsymbol{c} = \{\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \dots, \boldsymbol{c}^{(t)}, \dots\}$, where $t$ is the time step, BN regulate the input at each time step as follows,

$$\tilde{\boldsymbol{c}}_i^{(t)} = \frac{\boldsymbol{c}_i^{(t)} - \boldsymbol{\mu}_i}{\sqrt{\boldsymbol{\sigma}_i^2 + \epsilon}}, \tag{4}$$

where $\boldsymbol{c}_i^{(t)}$ is the input in $i$-th channel at $t$-th time step, $\boldsymbol{\mu}_i$ and $\boldsymbol{\sigma}_i$ are the mean and variance of input in channel dimension, and $\epsilon$ is a small constant to avoid denominator being zero. To ensure BN can represent the identity transformation, the normalized vector $\tilde{\boldsymbol{c}}_i^{(t)}$ is scaled and shifted in a learning manner as follows,

$$\mathrm{BN}(\boldsymbol{c}_i^{(t)}) = \boldsymbol{\lambda}_i \tilde{\boldsymbol{c}}_i^{(t)} + \boldsymbol{\beta}_i, \tag{5}$$

where $\boldsymbol{\lambda}_i$ and $\boldsymbol{\beta}_i$ are channel-wised learnable parameters.

## 4. Methodology

This section first introduces the specific form of membrane potential batch normalization. Then the re-parameterization technique that how to fold the MPBN into $V_{\text{th}}$ will be introduced in detail. Next, some key details for training the SNN and the pseudocode for the training and inference of our SNN will be given. Finally, we will provide plenty of ablation studies and the comparison of the loss landscape of the models with or without MPBN to show the effectiveness of the proposed method.

### 4.1. Membrane Potential Batch Normalization

As abovementioned, we argue that though the data flow has been normalized by the BN after the convolution layer, it will be disturbed again by the membrane potential updating operation. To better depict this, we give the vanilla form of LIF neuron with BN first as follows,

$$\boldsymbol{u}^{(t+1),\text{pre}} = \tau \boldsymbol{u}^{(t)} + \text{BN}(\mathbf{W}\boldsymbol{x}^{(t+1)}), \tag{6}$$

where $\tau$ is 0.25 in the paper following [21, 38, 4]. To regulate the disturbed data flow once again, We further embed another BN after the membrane potential updating operation, called MPBN. The LIF neuron with MPBN can be updated as

$$\tilde{\boldsymbol{u}}^{(t+1),\text{pre}} = \text{MPBN}(\boldsymbol{u}^{(t+1),\text{pre}}). \tag{7}$$

Obviously, $\boldsymbol{u}^{(t+1),\text{pre}}$ will be scaled and sifted, and some $\boldsymbol{u}^{(t+1),\text{pre}}$ less than $V_{\text{th}}$ may be greater than $V_{\text{th}}$ with MPBN and vice versa. This is abhorrent with the biology and MPBN will cause some extra computation burden in the inference compared with the vanilla one. To solve this problem, we also propose a training-inference-decoupled re-parameterization technique here.

### 4.2. Re-parameterization

With MPBN, the firing function will be updated as

$$\boldsymbol{o}^{(t+1)} = \begin{cases} 1 & \text{if MPBN}(\boldsymbol{u}^{(t+1),\text{pre}}) > V_{\text{th}} \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

If we unfold the MPBN, the above equation will be re-organized as

$$\boldsymbol{o}_i^{(t+1)} = \begin{cases} 1 & \text{if } \boldsymbol{\lambda}_i \frac{\boldsymbol{u}_i^{(t+1),\text{pre}} - \boldsymbol{\mu}_i}{\sqrt{\boldsymbol{\sigma}_i^2}} + \boldsymbol{\beta}_i > V_{\text{th}} \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

By folding the MPBN to $V_{\text{th}}$, the firing function will be further updated as

$$\boldsymbol{o}_i^{(t+1)} = \begin{cases} 1 & \text{if } \boldsymbol{u}^{(t+1),\text{pre}} > (\tilde{\boldsymbol{V}}_{\text{th}})_i \\ 0 & \text{otherwise} \end{cases}, \tag{10}$$
$$\text{where } (\tilde{\boldsymbol{V}}_{\text{th}})_i = \frac{(V_{\text{th}} - \boldsymbol{\beta}_i)\sqrt{\boldsymbol{\sigma}_i^2}}{\boldsymbol{\lambda}_i} + \boldsymbol{\mu}_i.$$

**Algorithm 1** Training and inference of our SNN.

**Training**

**Input**: An SNN to be trained with MPBN; training dataset; total training iteration: $I_{\text{train}}$.

**Output**: The well-trained SNN.

1: **for** all $i = 1, 2, \ldots, I_{\text{train}}$ iteration **do**
2: Get mini-batch training data, $\boldsymbol{x}_{\text{in}}(i)$ and class label, $\boldsymbol{y}(i)$;
3: Feed the $\boldsymbol{x}_{\text{in}}(i)$ into the SNN and regulate the data flow by BN and MPBN;
4: Calculate the SNN output, $\boldsymbol{o}_{\text{out}}(i)$ by Eq. 3 ;
5: Compute classification loss $L_{\text{CE}} = \mathcal{L}_{\text{CE}}(\boldsymbol{o}_{\text{out}}(i), \boldsymbol{y}(i))$;
6: Calculate the gradient w.r.t. $\mathbf{W}$ by Eq. 11;
7: Update $\mathbf{W}$: $(\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}})$ where $\eta$ is learning rate.
8: **end for**

**Re-parameterization**

**Input**: The trained SNN with MPBN; total number of MPBN: $n$.

**Output**: The re-parameterized trained SNN without MPBN but diverse $V_{\text{th}}$.

1: **for** all $i = 1, 2, \ldots, n$ number **do**
2: Fold the parameters of $i$-th MPBN into $i$-th $V_{\text{th}}$ by Eq. 10;
3: **end for**

**Inference**

**Input**: The re-parameterized trained SNN; test dataset; total test iteration: $I_{\text{test}}$.

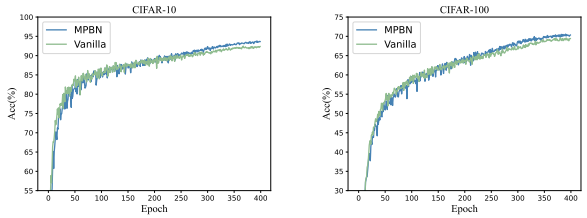
**Output**: The output.

1: **for** all $i = 1, 2, \ldots, I_{\text{test}}$ iteration **do**
2: Get mini-batch test data, $\boldsymbol{x}_{\text{in}}(i)$ and class label, $\boldsymbol{y}(i)$ in test dataset;
3: Feed the $\boldsymbol{x}_{\text{in}}(i)$ into the reparameterized SNN without MPBN;
4: Calculate the SNN output, $\boldsymbol{o}_{\text{out}}(i)$ by Eq. 3 ;
5: Compare the classification factor $\boldsymbol{o}_{\text{out}}(i)$ and $\boldsymbol{y}(i)$ for classification.
6: **end for**

It can be seen that by absorbing some parameters from MPBN, $V_{\text{th}}$ will be transformed to another channel-wised $(\tilde{\boldsymbol{V}}_{\text{th}})_i$. In this way, the extra computation burden caused by MPBN will be eliminated again in the inference time. Furthermore, the diversity of the spiking neuron will be improved with abundant firing parameters as the learnable firing threshold in other work [3, 49].

### 4.3. Training Framework

In the paper, the spatial-temporal backpropagation (STBP) algorithm [51] is adopted to train the SNN mod-




(a) CIFAR-10 (b) CIFAR-100

Figure 2: The accuracy curves of spiking ResNet20 with or without MPBN using 2 time steps on CIFAR-10 (left) and CIFAR-100 (right). The MPBN based SNNs obviously enjoy higher accuracy and easier convergence.

els. STBP treats the SNN model as a self-recurrent neural network thus enabling an error backpropagation mechanism following the same principles as in CNNs. However, there is still a problem impeding the direct training of SNNs. To demonstrate this problem, we formulate the gradient at the layer $l$ by the chain rule, given by

$$\frac{\partial L}{\partial \mathbf{W}^l} = \sum_t \left(\frac{\partial L}{\partial \boldsymbol{o}^{(t),l}} \frac{\partial \boldsymbol{o}^{(t),l}}{\partial \boldsymbol{u}^{(t),l}} + \frac{\partial L}{\partial \boldsymbol{u}^{(t+1),l}} \frac{\partial \boldsymbol{u}^{(t+1),l}}{\partial \boldsymbol{u}^{(t),l}}\right) \frac{\partial \boldsymbol{u}^{(t),l}}{\partial \mathbf{W}^l}, \tag{11}$$

where $\frac{\partial \boldsymbol{o}^{(t),l}}{\partial \boldsymbol{u}^{(t),l}}$ is the gradient of firing function at at $t$-th time step in $l$-th layer. Obviously, the non-differentiable firing activity of the spiking neuron will result in zero gradients everywhere, while infinity at $V_{\text{th}}$. Therefore, the gradient descent ($\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \frac{\partial L}{\partial \mathbf{W}^l}$) either freezes or updates to infinity in the backpropagation. To handle this problem, here, we also adopt the commonly used STE surrogate gradients as doing in other surrogate gradients (SG) methods [44, 20]. Mathematically, it is defined as:

$$\frac{d\boldsymbol{o}}{d\boldsymbol{u}} = \begin{cases} 1, & \text{if } 0 \leq \boldsymbol{u} \leq 1 \\ 0, & \text{otherwise} \end{cases}. \tag{12}$$

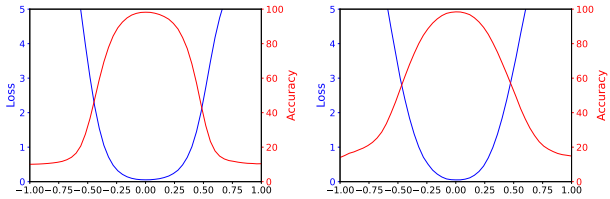
Then, the SNN model can be trained end-to-end. The training and inference of our SNN are detailed in Algo. 1.

### 4.4. Ablation Study

To verify the effectiveness of the MPBN, a lot of ablative studies using spiking ResNet20 architecture along with different time steps were conducted on the CIFAR-10 and CIFAR-100 datasets. The results of top-1 accuracy of these models are shown in Tab. 1. It's can be seen that the test accuracy of the SNNs with MPBN is always higher than these vanilla counterparts. For example, the accuracy of baseline SNN with 1 time step is 90.40%, while with MPBN, it will increase up to 92.22%, which is a huge improvement (more than 2.0%) in the SNN field. Moreover, we also show the test accuracy curves of ResNet20 using MPBN and its vanilla counterpart with 2 time steps on CIFAR-10/100 during training in Fig. 2. It can be observed obviously that the SNNs with MPBN also perform better on convergence speed. To sum up, the proposed MPBN can both improve accuracy and convergence speed, which are very important aspects in deep learning.

Table 1: Ablation experiments for MPBN.

| Dataset | Method | Time step | Accuracy |
|---|---|---|---|
| CIFAR-10 | baseline | 1 | 90.40% |
| | w/ MPBN | 1 | 92.22% |
| | baseline | 2 | 92.80% |
| | w/ MPBN | 2 | 93.54% |
| | baseline | 4 | 93.85% |
| | w/ MPBN | 4 | 94.28% |
| CIFAR-100 | baseline | 1 | 67.94% |
| | w/ MPBN | 1 | 68.36% |
| | baseline | 2 | 70.18% |
| | w/ MPBN | 2 | 70.79% |
| | baseline | 4 | 71.77% |
| | w/ MPBN | 4 | 72.30% |




(a) MPBN (b) Vanilla

Figure 3: The 1D loss landscape of spiking ResNet20 with and without MPBN.

### 4.5. Loss Landscape

We further inspect the 1D loss landscapes [36] of the SNNs with or without MPBN using spiking ResNet20 architecture in 2 time steps to show why the MPBN can improve accuracy and convergence speed in Fig. 3. It can be observed that the loss landscape of the SNN model with MPBN is flatter than that of the SNN without MPBN. This indicates that the MPBN makes the landscape of the corresponding optimization problem smoother [36], thus making the gradients more predictive and network faster convergence. The results here provide convincing evidence for ablation studies in Section 4.4.

## 5. Experiments

In this section, abundant experiments were conducted to verify the effectiveness of the MPBN using widely-used spiking ResNet20 [44, 46], VGG16 [44], ResNet18 [11],

Table 2: Comparison with SoTA methods on CIFAR-10.

| Dataset | Method | Type | Architecture | Timestep | Accuracy |
|---|---|---|---|---|---|
| CIFAR-10 | SpikeNorm [46] | ANN2SNN | VGG16 | 2500 | 91.55% |
| | Hybrid-Train [45] | Hybrid training | VGG16 | 200 | 92.02% |
| | Spike-basedBP [34] | SNN training | ResNet11 | 100 | 90.95% |
| | Joint A-SNN [19] | SNN training | ResNet18 | 4 | 95.45% |
| | GLIF [60] | SNN training | ResNet19 | 2 | 94.44% |
| | PLIF [12] | SNN training | PLIFNet | 8 | 93.50% |
| | Diet-SNN [44] | SNN training | VGG16 | 5 | 92.70% |
| | | | | 10 | 93.44% |
| | | | ResNet20 | 5 | 91.78% |
| | | | | 10 | 92.54% |
| | RecDis-SNN [20] | SNN training | ResNet19 | 2 | 93.64% |
| | | | | 4 | 95.53% |
| | | | | 6 | 95.55% |
| | Dspike [38] | SNN training | ResNet20 | 2 | 93.13% |
| | | | | 4 | 93.66% |
| | | | | 6 | 94.25% |
| | STBP-tdBN [62] | SNN training | ResNet19 | 2 | 92.34% |
| | | | | 4 | 92.92% |
| | | | | 6 | 93.16% |
| | TET [7] | SNN training | ResNet19 | 2 | 94.16% |
| | | | | 4 | 94.44% |
| | | | | 6 | 94.50% |
| | Real Spike [21] | SNN training | ResNet19 | 2 | 95.31% |
| | | | | 4 | 95.51% |
| | | | | 6 | 96.10% |
| | InfLoR-SNN [16] | SNN training | ResNet20 | 5 | 93.01% |
| | | | | 10 | 93.65% |
| | **MPBN** | SNN training | ResNet19 | 1 | **96.06%**±0.10 |
| | | | | 2 | **96.47%**±0.08 |
| | | | ResNet20 | 1 | **92.22%**±0.11 |
| | | | | 2 | **93.54%**±0.09 |
| | | | | 4 | **94.28%**±0.07 |
| | | | VGG16 | 2 | **93.96%**±0.09 |
| | | | | 4 | **94.44%**±0.08 |

ResNet19 [62], and ResNet34 [11] on both static datasets including CIFAR-10 [32], CIFAR-100 [32], and ImageNet [6], and one neuromorphic dataset, CIFAR10-DVS [35]. The specific introduction for these datasets has be detailed in many recent works [62, 44, 21, 38]. Here, we mainly introduce these hyper-parameters and data preprocessing in detail. We used the widely adopted LIF neuron in our SNN models as other works about direct training methods [44, 46]. These hyper-parameters for LIF neuron about the initial firing threshold $V_{\text{th}}$ and the membrane potential decaying constant $\tau_{\text{decay}}$ are $0.5$ and $0.25$ respectively. For static image datasets, since encoding the 8-bit RGB images into 1-bit spikes will lose too much information, we use an ANN-like convolutional layer and a LIF layer to encode the images to spikes for all the rest of the layers, as in recent works [62, 44, 21, 38].

### 5.1. Comparison with SoTA Methods

**CIFAR-10.** On CIFAR-10, we trained our SNN model using the SGD optimizer with 0.9 momentum. The initial learning rate is 0.1 and decays to 0 in cosine form. The total training time is 400 epochs. To fairly compare with these recent SoTA methods [38, 20, 16], we also adopt data normalization, random horizontal flipping, cropping, and cutout [8] for data augmentation. We run three times for each experiment to report the "mean ± std" in Tab. 2. It can be seen that our models can outperform other methods over all these chosen widely adopted architectures with

Table 3: Comparison with SoTA methods on CIFAR-100.

| Dataset | Method | Type | Architecture | Timestep | Accuracy |
|---|---|---|---|---|---|
| CIFAR-100 | SpikeNorm [46] | ANN2SNN | ResNet20 | 2500 | 64.09% |
| | RMP [23] | ANN2SNN | ResNet20 | 2048 | 67.82% |
| | Hybrid-Train [45] | Hybrid training | VGG11 | 125 | 67.90% |
| | IM-Loss [15] | SNN training | VGG16 | 5 | 70.18% |
| | Joint A-SNN [19] | SNN training | ResNet18 | 4 | 77.39% |
| | | | ResNet34 | 4 | 79.76% |
| | Dspike [38] | SNN training | ResNet20 | 2 | 71.68% |
| | | | | 4 | 73.35% |
| | | | | 6 | 74.24% |
| | TET [7] | SNN training | ResNet19 | 2 | 72.87% |
| | | | | 4 | 74.47% |
| | | | | 6 | 74.72% |
| | RecDis-SNN [20] | SNN training | ResNet19 | 4 | 74.10% |
| | | | VGG16 | 5 | 69.88% |
| | InfLoR-SNN [16] | SNN training | ResNet20 | 5 | 71.19% |
| | | | VGG16 | 5 | 71.56% |
| | Real Spike [21] | SNN training | ResNet20 | 5 | 66.60% |
| | | | VGG16 | 5 | 70.62% |
| | GLIF [60] | SNN training | ResNet19 | 2 | 75.48% |
| | | | | 4 | 77.05% |
| | TEBN [10] | SNN training | ResNet19 | 2 | 75.86% |
| | | | | 4 | 76.13% |
| | | | | 6 | 76.41% |
| | **MPBN** | SNN training | VGG16 | 4 | **74.74%**±0.11 |
| | | | ResNet19 | 1 | **78.71%**±0.10 |
| | | | | 2 | **79.51%**±0.07 |
| | | | ResNet20 | 2 | **70.79%**±0.08 |
| | | | | 4 | **72.30%**±0.08 |

Table 4: Comparison with SoTA methods on ImageNet.

| Dataset | Method | Type | Architecture | Timestep | Accuracy |
|---|---|---|---|---|---|
| ImageNet | STBP-tdBN [62] | SNN training | ResNet34 | 6 | 63.72% |
| | TET [7] | SNN training | ResNet34 | 6 | 64.79% |
| | MS-ResNet [27] | SNN training | ResNet18 | 6 | 63.10% |
| | OTTT [54] | SNN training | ResNet34 | 6 | 63.10% |
| | Real Spike [21] | SNN training | ResNet18 | 4 | 63.68% |
| | SEW ResNet [11] | SNN training | ResNet18 | 4 | 63.18% |
| | | | ResNet34 | 4 | 67.04% |
| | **MPBN** | SNN training | ResNet18 | 4 | **63.14%**±0.08 |
| | | | ResNet34 | 4 | **64.71%**±0.09 |

fewer time steps. For example, The accuracy of spiking ResNet19 trained with MPBN with only 1 time step can be up to 96.06%, while the Real Spike [21] needs 6 time steps to reach a comparable result and the RecDis-SNN [61] even still underperforms 0.51% with 6 time steps. this superiority can also be observed in the results regarding the spiking ResNet20 and VGG16.

**CIFAR-100.** For CIFAR-100, we adopted the same settings as in CIFAR-10. The proposed MPBN also performs well on CIFAR-100. It can be seen that our method gets the best accuracy over all these networks even with fewer time steps. For instants, the ResNet19 trained with MPBN

Table 5: Comparison with SoTA methods on CIFAR10-DVS.

| Dataset | Method | Type | Architecture | Timestep | Accuracy |
|---|---|---|---|---|---|
| CIFAR10-DVS | Rollout [33] | Rollout | DenseNet | 10 | 66.80% |
| | LIAF-Net [53] | Conv3D | LIAF-Net | 10 | 71.70% |
| | LIAF-Net [53] | LIAF | LIAF-Net | 10 | 70.40% |
| | STBP-tdBN [62] | SNN training | ResNet19 | 10 | 67.80% |
| | RecDis-SNN [20] | SNN training | ResNet19 | 10 | 72.42% |
| | Real Spike [21] | SNN training | ResNet19 | 10 | 72.85% |
| | | | ResNet20 | 10 | 78.00% |
| | **MPBN** | SNN training | ResNet19 | 10 | **74.40%**±0.20 |
| | | | ResNet20 | 10 | **78.70%**±0.10 |

can achieve 78.71% top-1 accuracy with only 1 time step, which outperforms other SoTA methods such as TET, GLIF, TEBN, and RecDis-SNN even with 4 or 6 time steps about 1.66%-3.99% .

**ImageNet.** On ImageNet, we used standard data normalization, random horizontal flipping, and cropping for data augmentation and trained the networks for 320 epochs as in [11]. The optimizer setting also keeps the same with CIFAR datasets. The results for ImageNet are presented in Tab. 4. It can be seen that the accuracy of our method is better than that of these recent SoTA methods, only relatively smaller compared with SEW ResNet [11] for spiking ResNet34. However, SEW ResNet is not a typical SNN model. It adopts the activation before addition form-based ResNet and its blocks will fire positive integer spikes. In this way, the event-driven and multiplication-addition transform advantages of SNNs will be lost. While we adopt the original ResNet, which fires standard binary spikes.

**CIFAR10-DVS.** We also adopted the neuromorphic dataset, CIFAR10-DVS in the paper to verify the effectiveness of the MPBN. We also split the dataset into 9K training images and 1K test images, and resize them to $48 \times 48$ for data augmentation as in [51, 21]. The learning rate is 0.01 and other settings are the same as CIFAR-10. It can be seen that the MPBN also shows superiority in this dataset.

### 5.2. Extension of the MPBN

In CNNs, the most widely used BN is channel-wised. This is because element-wised BNs are very time-consuming and can not be folded into weights, otherwise, the channel-wised weight-sharing mechanism will be destroyed. However, the MPBN adopts the firing threshold-folded manner and the firing threshold need not keep the same along the channels, therefore, MPBN can use the element-wised form freely. In this way, $V_{\text{th}}$ will be transformed to element-wised ones as follows,

$$(\tilde{\boldsymbol{V}}_{\text{th}})_{i,j,k} = \frac{(V_{\text{th}} - \boldsymbol{\beta}_{i,j,k})\sqrt{\boldsymbol{\sigma}^2_{i,j,k}}}{\boldsymbol{\lambda}_{i,j,k}} + \boldsymbol{\mu}_{i,j,k}, \qquad (13)$$

Table 6: Comparison with learnable firing threshold methods.

| Dataset | Method | Time step | Accuracy |
|---|---|---|---|
| CIFAR-10 | channel-wised | 4 | 94.28% |
| | element-wised | 4 | 94.42% |
| CIFAR-100 | channel-wised | 4 | 72.30% |
| | element-wised | 4 | 72.49% |

where$(\tilde{\boldsymbol{V}}_{\text{th}})_{i,j,k}$ is the transformed firing threshold of the neuron comes from $i$-th channel in the spatial position $(j, k)$. To investigate the performance of the element-wised MPBN, here we also provide a comparison of the vanilla MPBN and its extension. The results of top-1 accuracy of the spiking ResNet20 with 4 time steps on CIFAR datasets are shown in Tab. 6. Though the two versions all perform well, the element-wised MPBN is relatively better than the channel-wised MPBN. This may be because element-wised MPBN can learn more firing threshold values, which means a richer representation ability for SNNs.

## 6. Conclusion

In the paper, we advocated adding the MPBN before the firing function to regulate the disturbed data flow again. We also provided a training-inference-decoupled re-parameterization technique to fold the trained MPBN into the firing threshold to eliminate the extra time burden induced by MPBN in the inference time. Furthermore, the channel-wised and element-wised MPBN in different granularities were explored. Extensive experiments verified that the proposed MPBN can consistently achieve good performance.

## Acknowledgment

This work is supported by grants from the National Natural Science Foundation of China under contracts No.12202412 and No.12202413.

## References


[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente Kuang, Rajit Manohar, W.P. Risk, Bryan Jackson, and Dharmendra Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34:1537–1557, 10 2015. 1

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2

[3] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons, 2018. 4

[4] Xiang Cheng, Yunzhe Hao, Jiaming Xu, and Bo Xu. Lisnn: Improving spiking neural networks with lateral interactions for robust object recognition. pages 1519–1525, 07 2020. 4

[5] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Andrew Lines, Andreas Wild, Hong Wang, and Deepak Mathaikutty. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38:82 – 99, 01 2018. 1

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009. 6

[7] Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *International Conference on Learning Representations*, 2022. 6, 7

[8] Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout, 2017. 6

[9] Travis DeWolf. Spiking neural networks take control. *Science Robotics*, 6(58):eabk3268, 2021. 1

[10] Chaoteng Duan, Jianhao Ding, Shiyan Chen, Zhaofei Yu, and Tiejun Huang. Temporal effective batch normalization in spiking neural networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 1, 3, 7

[11] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, and Yonghong Tian. Deep residual learning in spiking neural networks. 2021. 2, 5, 6, 7, 8

[12] Wei Fang, Zhaofei yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. 08 2021. 1, 3, 6

[13] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *Int. J. Neural Syst.*, 19:295–308, 08 2009. 1

[14] Yufei Guo and Yuanpei Chen. Neuroclip: Neuromorphic data understanding by clip and snn. *arXiv preprint arXiv:2306.12073*, 2023. 1

[15] Yufei Guo, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Yinglei Wang, Xuhui Huang, and Zhe Ma. IM-loss: Information maximization loss for spiking neural networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 7

[16] Yufei Guo, Yuanpei Chen, Liwen Zhang, YingLei Wang, Xiaode Liu, Xinyi Tong, Yuanyuan Ou, Xuhui Huang, and Zhe Ma. Reducing information loss for spiking neural networks. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pages 36–52. Springer, 2022. 6, 7

[17] Yufei Guo, Xuhui Huang, and Zhe Ma. Direct learning-based deep spiking neural networks: a review. *Frontiers in Neuroscience*, 17:1209795, 2023. 1, 2

[18] Yufei Guo, Xiaode Liu, Yuanpei Chen, Liwen Zhang, Weihang Peng, Yuhan Zhang, Xuhui Huang, and Zhe Ma. Rmploss: Regularizing membrane potential distribution for spiking neural networks. *arXiv preprint arXiv:2308.06787*, 2023. 2

[19] Yufei Guo, Weihang Peng, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Xuhui Huang, and Zhe Ma. Joint a-snn: Joint training of artificial and spiking neural networks via self-distillation and weight factorization. *Pattern Recognition*, page 109639, 2023. 1, 2, 6, 7

[20] Yufei Guo, Xinyi Tong, Yuanpei Chen, Liwen Zhang, Xiaode Liu, Zhe Ma, and Xuhui Huang. Recdis-snn: Rectifying membrane potential distribution for directly training spiking neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 326–335, June 2022. 2, 3, 5, 6, 7, 8

[21] Yufei Guo, Liwen Zhang, Yuanpei Chen, Xinyi Tong, Xiaode Liu, YingLei Wang, Xuhui Huang, and Zhe Ma. Real spike: Learning real-valued spikes for spiking neural networks. In *European Conference on Computer Vision*, pages 52–68. Springer, 2022. 1, 3, 4, 6, 7, 8

[22] Bing Han and Kaushik Roy. Deep spiking neural network: Energy efficiency through time based coding. In *ECCV*, pages 388–404. Springer, 2020. 2

[23] Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. pages 13555–13564, 06 2020. 7

[24] Yunzhe Hao, Xuhui Huang, Meng Dong, and Bo Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule. *Neural Networks*, 121, 09 2019. 2

[25] Zecheng Hao, Tong Bu, Jianhao Ding, Tiejun Huang, and Zhaofei Yu. Reducing ann-snn conversion error through residual membrane potential, 2023. 2

[26] Zecheng Hao, Jianhao Ding, Tong Bu, Tiejun Huang, and Zhaofei Yu. Bridging the gap between anns and snns by calibrating offset spikes, 2023. 2

[27] Yifan Hu, Yujie Wu, Lei Deng, and Guoqi Li. Advancing residual learning towards powerful deep spiking neural networks. *arXiv preprint arXiv:2112.08954*, 2021. 7

[28] Shin-ichi Ikegawa, Ryuji Saiin, Yoshihide Sawada, and Naotake Natori. Rethinking the role of normalization and residual blocks for spiking neural networks. *Sensors*, 22(8), 2022. 1, 3

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 2

[30] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11270–11277, 2020. 1

[31] Y. Kim and P. Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch, 2020. 1, 3

[32] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 6

[33] Alexander Kugele, Thomas Pfeil, Michael Pfeiffer, and Elisabetta Chicca. Efficient processing of spatio-temporal data streams with spiking neural networks. *Frontiers in Neuroscience*, 14:439, 05 2020. 8

[34] Chankyu Lee, Syed Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14:119, 02 2020. 6

[35] Hongmin Li. Cifar10-dvs: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11, 05 2017. 6

[36] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018. 5

[37] Yuhang Li, Shikuang Deng, Xin Dong, and et al. A free lunch from ann: Towards efficient, accurate spiking neural networks calibration. In *ICML*, 2021. 2

[38] Yuhang Li, Yufei Guo, Shanghang Zhang, Shikuang Deng, Yongqing Hai, and Shi Gu. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34, 2021. 2, 4, 6, 7

[39] S. A. Lobov and et al. Spatial properties of stdp in a spiking neural network enable controlling a mobile robot. *Frontiers in Neuroscience*, 2020. 2

[40] Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable learning-to-normalize via switchable normalization, 2019. 2

[41] Mitchell A Nahmias, Bhavin J Shastri, Alexander N Tait, and Paul R Prucnal. A leaky integrate-and-fire laser neuron for ultrafast cognitive computing. *IEEE journal of selected topics in quantum electronics*, 19(5):1–12, 2013. 3

[42] Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, Feng Chen, Ning Deng, Si Wu, Yu Wang, Yujie Wu, Zheyu Yang, Cheng Ma, Guoqi Li, Wentao Han, and L.P. Shi. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572:106, 08 2019. 1

[43] Peter, U., Diehl, Matthew, and Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015. 2

[44] Nitin Rathi and Kaushik Roy. DIET-SNN: direct input encoding with leakage and threshold optimization in deep spiking neural networks. *CoRR*, abs/2008.03658, 2020. 5, 6

[45] Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. 05 2020. 6, 7

[46] 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, 02 2018. 2, 5, 6, 7

[47] Jiangrong Shen, Qi Xu, Jian K Liu, Yueming Wang, Gang Pan, and Huajin Tang. Esl-snns: An evolutionary structure learning strategy for spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 86–93, 2023. 1

[48] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017. 2

[49] Siqi Wang, Tee Hiang Cheng, and Meng-Hiot Lim. LTMD: Learning improvement of spiking neural networks with learnable thresholding neurons and moderate dropout. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 4

[50] Yujie Wu, Lei Deng, and et al. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 2018. 2

[51] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and L.P. Shi. Direct training for spiking neural networks: Faster, larger, better. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1311–1318, 07 2019. 1, 2, 4, 8

[52] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 2

[53] Zhenzhi Wu, Hehui Zhang, Yihan Lin, Guoqi Li, Meng Wang, and Ye Tang. Liaf-net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6249–6262, 2022. 8

[54] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. *arXiv preprint arXiv:2210.04195*, 2022. 7

[55] Rong Xiao, Yu Wan, Baosong Yang, Haibo Zhang, Huajin Tang, Derek F Wong, and Boxing Chen. Towards energy-preserving natural language understanding with spiking neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:439–447, 2022. 1

[56] Qi Xu, Yaxin Li, Xuanye Fang, Jiangrong Shen, Jian K Liu, Huajin Tang, and Gang Pan. Biologically inspired structure learning with reverse knowledge distillation for spiking neural networks. *arXiv preprint arXiv:2304.09500*, 2023. 1

[57] Qi Xu, Yaxin Li, Jiangrong Shen, Jian K Liu, Huajin Tang, and Gang Pan. Constructing deep spiking neural networks from artificial neural networks with knowledge distillation. *arXiv preprint arXiv:2304.05627*, 2023. 1

[58] Qi Xu, Yaxin Li, Jiangrong Shen, Pingping Zhang, Jian K Liu, Huajin Tang, and Gang Pan. Hierarchical spiking-based

model for efficient image classification with enhanced feature extraction and encoding. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 1

[59] Qi Xu, Jiangrong Shen, Xuming Ran, Huajin Tang, Gang Pan, and Jian K Liu. Robust transcoding sensory information with neural spikes. *IEEE Transactions on Neural Networks and Learning Systems*, 33(5):1935–1946, 2021. 1

[60] Xingting Yao, Fanrong Li, Zitao Mo, and Jian Cheng. Glif: A unified gated leaky integrate-and-fire neuron for spiking neural networks. *arXiv preprint arXiv:2210.13768*, 2022. 6, 7

[61] Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. 02 2020. 7

[62] Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. 10 2020. 1, 3, 6, 7, 8