Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks

Jianhao Ding¹, Zhaofei Yu¹,²,³*, Yonghong Tian¹,³* and Tiejun Huang¹,²,³

¹Department of Computer Science and Technology, Peking University
²Institute for Artificial Intelligence, Peking University
³Peng Cheng Laboratory
djh01998@stu.pku.edu.cn, {yuzf12,yhtian,tjhuang}@pku.edu.cn

Abstract

Spiking Neural Networks (SNNs), as bio-inspired energy-efficient neural networks, have attracted great attentions from researchers and industry. The most efficient way to train deep SNNs is through ANN-SNN conversion. However, the conversion usually suffers from accuracy loss and long inference time, which impede the practical application of SNN. In this paper, we theoretically analyze ANN-SNN conversion and derive sufficient conditions of the optimal conversion. To better correlate ANN-SNN and get greater accuracy, we propose Rate Norm Layer to replace the ReLU activation function in source ANN training, enabling direct conversion from a trained ANN to an SNN. Moreover, we propose an optimal fit curve to quantify the fit between the activation value of source ANN and the actual firing rate of target SNN. We show that the inference time can be reduced by optimizing the upper bound of the fit curve in the revised ANN to achieve fast inference. Our theory can explain the existing work on fast reasoning and get better results. The experimental results show that the proposed method achieves near lossless conversion with VGG-16, PreActResNet-18, and deeper structures. Moreover, it can reach 8.6× faster reasoning performance under 0.265× energy consumption of the typical method. The code is available at https://github.com/DingJianhao/OptSNNConversion-RNL-RIL.

1 Introduction

As a representative of artificial intelligence methods, deep learning has begun to exceed or approach human performance in various tasks, including image classification, natural language processing, and electronic sports [He et al., 2016a; Brown et al., 2020; Berner et al., 2019]. But this success is at the cost of high energy consumption. Recently, neuromorphic hardware, including TrueNorth, SpiNNaker, Loihi, and so on [DeBole et al., 2019; Painkras et al., 2013; Davies et al., 2018], is attracting more and more researchers due to their high temporal resolution and low power budget. This kind of hardware runs Spiking Neural Networks (SNNs) instead of Artificial Neural Networks (ANNs). With unique memory and communication designs, an SNN implemented on SpiNNaker can achieve the power consumption of 0.3W for MNIST classification [Stromatias et al., 2015]. For object detection, the spiking version of a YOLO model is estimated to be at least 100 times more energy-efficient than that on GPU [Kim et al., 2020]. Such dedicated hardware and algorithms are highly appealing for mobile applications like autonomous vehicles and other power-limited scenes.

Nevertheless, training high-performance SNNs is a non-trivial problem. The neurons in SNNs emit discrete spikes, which disables the direct backpropagation training. Up to now, the training algorithms for SNNs can be summarised into four methodologies: supervised backpropagation through time [Wu et al., 2019; Zenke and Vogels, 2021], unsupervised STDP learning [Kheradpisheh et al., 2018; Diehl and Cook, 2015], ANN-SNN conversion [Cao et al., 2015; Diehl et al., 2015; Rueckauer et al., 2017], and other mixture methods [Lee et al., 2018; Tavanaei and Maida, 2019; Rathi et al., 2020]. For deep SNN training, ANN-SNN conversion requires less GPU computing than supervised training with surrogate gradients. Meanwhile, it has yielded the best performance in large-scale networks and datasets among methodologies. Therefore, ANN-SNN conversion has become the first choice for deep SNN training, which is also the focus of this paper.

As illustrated in Fig. 1, ANN-SNN conversion is to map the parameters of a pre-trained ANN to an SNN with low accuracy loss. Cao et al. [2015] started the study of ANN-SNN conversion. They found the equivalence between the ReLU activation and the spiking neurons’ firing rate, which
is the foundation of later rate-based methods. Diehl et al. [2015] attributed the performance loss to inappropriate activation of neurons and proposed Weight Normalization methods (model-based and data-based) to scale the ANN weights. Rueckauer et al. [2017] gave a detailed theoretical explanation of ANN-SNN conversion and proposed a new reset-by-subtraction neuron to overcome accuracy degradation. They also extended the use of bias and Batch Normalization (BN) and proposed the Max Normalization algorithm (Max Norm for short), which uses maximum values of activation as scaling factors. Kim et al. [2020] suggested to use channel-wise normalization for convolutional nets. Different from conversion using ANN activation, Sengupta et al. [2019] proposed SpikeNorm, which makes use of spiking statistics to set the thresholds. To enable more accurate conversion, Rueckauer et al. [2017] further proposed Robust Normalization (Robust Norm for short), where the scaling factor changes from maximum activation value to 99.9% of activation. Yet this is not the first attempt to manually manipulate the threshold or factor. Cao et al. [2015] set the firing thresholds based on spike density. These practices allow the firing rates of some neurons to be constant 1. Here we refer to this phenomenon as spike saturation. However, both the maximum and 99.9% are rigid, which inspires us to explore a trainable way to achieve low conversion loss.

The conversion methods mentioned above all incur long simulation time when applied to deeper networks and more complicated datasets. That is, converted SNNs need a longer time to rival the original ANNs in precision. This restricts the practical promotion, such as real-time tracking and detection. Robust Normalization somewhat mitigates this problem by increasing the firing rates. Spike saturation actually causes the subsequent layers to infer faster. Based on this observation, Han et al. [2020] started to improve the inference latency by scaling the SpikeNorm thresholds. Han et al. then gave a theoretical analysis of the scale (setting the firing threshold as the expectation of weights × spikes), but they used the manually set value eventually. Nevertheless, this inspires us that inference latency and parameters can establish associations on the model. A hybrid training scheme also helps. Rathi et al. [2020] realized fewer inference time-steps by conversion-based initialization and spiking-dependence backpropagation. Other methods concern coding schemes to achieve fast inference (i.e., shorter inference time), including Temporal-Switch Coding [Han and Roy, 2020], FS-conversion coding [Stöckl and Maass, 2021]. However, simply bypassing rate coding is not that rational. Though rate coding is not the perfect coding scheme, it is partially in line with observations in the visual cortex [Rullen and Thorpe, 2001]. Therefore, fast inference for rate-encoding deep SNNs is an important research direction. But it still lacks instructive principles and theories.

In this paper, we propose an ANN-SNN conversion method that enables high accuracy and low latency. The main contributions of this paper are summarized as follows:

- We theoretically analyze ANN-SNN conversion and derive the sufficient conditions of the optimal conversion. Based on this, we propose Rate Norm Layer to replace the ReLU activation function in source ANN, enabling direct conversion from a trained ANN to an SNN. This will reduce the potential loss of information caused by normalization.

- We propose an optimal fit curve to quantify the fit between the activation value of source ANN and the actual firing rate of target SNN, and derive one upper bound of this convergent curve. We show that based on the Squeeze Theorem, the inference time can be reduced by optimizing the coefficient in the upper bound. These results can not only systematically explain previous findings that reasonable scaling of the threshold can speed up inference, but also give a proper theoretical basis for fast inference research.

- We demonstrate the utility of the proposed method with near loss-less conversion in deep network architectures on the MNIST, CIFAR-10, CIFAR-100 datasets. Moreover, it achieves 8.6× faster reasoning under 0.265× energy consumption of the typical method.

2 Methods

In this section, a theory for ANN-SNN conversion is first introduced. Based on this, Rate Norm Layer with trainable threshold is thus proposed. Then, we analyse the reason for slow inference and suggest optimization for fit of firing rates. Finally, we present a stage-wise learning strategy for accurate and fast SNN.

2.1 Theory for Conversion from ANN to SNN

The fundamental principle of ANN-SNN conversion is to match analog neurons’ activation with spiking neurons’ firing rate. One common way is to convert ReLU nonlinearity activation to the Integrate-and-Fire (I&F) neuron. To be specific, for analog neurons in layer $l$ ($l = 1, 2, ..., L$), the ReLU activation can be described by:

$$a_l = \max(W_{l-1} a_{l-1} + b_{l-1}, 0),$$  \hspace{1cm} (1)

where vector $a_l$ is the output of all ReLU-based artificial neurons in layer $l$, $W_{l-1}$ and $b_{l-1}$ is the weight and the bias term for the neurons in layer $l − 1$.

As for the I&F neuron, the membrane potential $v_i^l(t)$ for the $i$-th neuron in layer $l$ is formulated by:

$$\frac{dv_i^l(t)}{dt} = \sum_{j} \sum_{t_j \in T_j} W_{l-1}^{ij} \delta(t - t_j) + b_{l-1}^i,$$  \hspace{1cm} (2)

where $b_{l-1}^i$ denotes the bias current to the $i$-th neuron, $W_{l-1}^{ij}$ denotes the synaptic weight between the $j$-th presynaptic neuron in layer $l − 1$ and the $i$-th neuron in layer $l$. $\delta(\cdot)$ is the delta function. $T_j$ denotes the set of spike time of the $j$-th presynaptic neuron, i.e., $T_j = \{t_j^{(1)}, t_j^{(2)}, ..., t_j^{(K)}\}$. When the membrane potential $v_i^l(t)$ exceeds the firing threshold $v_{th,l}$ in layer $l$, a spike is generated and the membrane potential $v_i^l(t)$ is reset to the rest value $v_{rest} < v_{th,l}$.

To match analog neurons’ activation with spiking neurons’ firing rate, we discretize and vectorize Eq. 2 into time-steps
and obtain the spiking neuron model for layer \(l\).

\[
\begin{align*}
\mathbf{m}_l(t) &= \mathbf{v}_l(t-1) + W_1^{-1} s_{l-1}(t) + \mathbf{b}_{l-1}, \\
s_{l}(t) &= U(\mathbf{m}_l(t) - v_{th,l}), \\
v_{l}(t) &= \mathbf{m}_l(t) - v_{th,l}s_{l}(t),
\end{align*}
\]  

(3)

where \(\mathbf{m}_l(t)\) and \(v_l(t)\) represent the membrane potential of all I&F neurons in layer \(l\) after neuronal dynamics and after the trigger of a spike at time \(t\), \(U(\cdot)\) is the Heaviside Step Function, \(s_l(t)\) denotes the vector of binary spikes, the element in which equals 1 if there is a spike and 0 otherwise. \(\mathbf{b}_{l-1}\) is the vector of \(b_{l-1}\), and \(W_1^{-1}\) is the weight matrix. Note that here we use the "soft reset" [Han et al., 2020] instead of the "hard reset". At the moment of a spike, the membrane potential \(v_l(t)\) is reduced by an amount equal to the firing threshold \(v_{th,l}\), instead of going back to the reset value.

Based on these definitions, we can derive the relationship between the firing rate \(r_l(t)\) of spiking neurons in layer \(l\) and \(r_{l-1}(t)\) of neurons in layer \(l-1\), which is depicted in Lemma 1. The proof can be found in the Appendix.

**Lemma 1.** For a spiking neural network consisting of the reset-by-subtraction neurons mentioned in Eq. 3, assume that \(W_{l-1}\) and \(b_{l-1}\) are the parameters for layer \(l-1\). Then when \(t \to \infty\), the relation of the firing rate \(r_l(t)\) and \(r_{l-1}(t)\) is given by:

\[
r_l = \text{clip}\left(\frac{W_{l-1}r_{l-1} + b_{l-1}}{v_{th,l}}, 0, 1\right),
\]

(4)

where \(\text{clip}(x,0,1) = x \text{ when } x \in [0,1], \text{ clip}(x,0,1) = 1 \text{ when } x > 1, \text{ and clip}(x,0,1) = 0 \text{ when } x < 0.\)

With Lemma 1 and Eq. 1, We can derive the theorem for conversion from ANN to SNN:

**Theorem 1.** For an \(L\)-layered ANN with the ReLU activation and an \(L\)-layered SNN with the reset-by-subtraction neurons, assume that \(W^{\text{ANN}}_{l-1}, b^{\text{ANN}}_{l-1}\) are the parameters for layer \(l-1\) of the ANN, and \(W^{\text{SNN}}_{l-1}, b^{\text{SNN}}_{l-1}\) are the parameters for layer \(l-1\) of the SNN. \(\text{max}_l\) is the maximum activation of layer \(l\) in ANN, and \(v_{th,l}\) is the firing threshold of layer \(l\) in SNN. The ANN can be converted to the SNN when \(t \to \infty\) (Eq. 1 equals Eq. 4) if for \(l = 1, 2, ..., L\), the following equations hold:

\[
\frac{W^{\text{SNN}}_{l-1}}{v_{th,l}} = \frac{W^{\text{ANN}}_{l-1}\text{max}_{l-1}}{\text{max}_l}, \quad \frac{b^{\text{SNN}}_{l-1}}{v_{th,l}} = \frac{b^{\text{ANN}}_{l-1}}{\text{max}_l}.
\]

(5)

The proof of Theorem 1 is presented in the Appendix. Eq. 5 implies that scaling operations are necessary to convert ANN to SNN, either scaling weights (i.e. weight normalization) or setting thresholds (i.e. threshold balancing). In this sense, weight normalization is equivalent to threshold balancing.

### 2.2 Rate Norm Layer

The choices of scaling factors are often empirical, and post-training [Rueckauer et al., 2016; Han et al., 2020]. To overcome this, we propose Rate Norm Layer (RNL) to replace the ReLU activation in ANN. The idea is to use a clip function with a trainable upper bound to output the simulated firing rate, which is the limitation of actual firing rate in SNN when inference time \(T \to \infty\). Here we denote the simulated firing rate as \(\hat{r}_l\). Rate Norm Layer can be formally expressed as follows:

\[
\begin{align*}
\theta_l &= p_l \cdot \max(W_{l-1}\hat{r}_{l-1} + b_{l-1}), \\
z_l &= \text{clip}(W_{l-1}\hat{r}_{l-1} + b_{l-1}, 0, \theta_l), \\
\hat{r}_l &= \frac{z_l}{\theta_l},
\end{align*}
\]

(6)

where \(p_l\) is a trainable scalar \((p_l \in [0,1])\), and \(\theta_l\) is the threshold of the \(l\)-th layer. With Theorem 1 satisfied \((v_{th,l} = \theta_l)\) and \(p_l = 1\), one can find that Eq. 6 is equivalent to Eq. 4. In this case, RNL will degenerate to the Max Norm algorithm, which scales the weight \(W_{l-1}\) by \(\max_{l-1}\) and the bias \(b_{l-1}\) by \(\frac{1}{\max_{l-1}}\). A diagram comparison of different scaling schemes can be seen in Fig. 2. For mini-batch training, different batches have different maximum outputs. To reduce the perturbation caused by data sampling, \(\text{running}\max(W_{l-1}\hat{r}_{l-1} + b_{l-1})\) is used instead of \(\max(W_{l-1}\hat{r}_{l-1} + b_{l-1})\). The design of the Rate Norm Layer mainly considers the following three factors:

- Compared with directly cutting the simulated firing rate to 1, the backpropagation becomes more effective in RNL training. \(\max(W_{l-1}\hat{r}_{l-1} + b_{l-1})\) and \(\text{running}\max(W_{l-1}\hat{r}_{l-1} + b_{l-1})\) can enable the gradient to flow out smoothly. Their participation in threshold calculating is similar to Batch Norm (BN) in mini-batch training. However, RNL cannot replace BN, because BN rescales the data to a normal distribution, which is related to the characteristics of ANN.

- The threshold \(\theta_l\) enables better generalization. Existing solutions mainly focus on using a subset of the training data for offline normalization. This will potentially influence the generalization of the scaled SNN for data out of subset. In contrast, \(\theta_l\) uses all data in training, which can be used directly in SNN inference.

- The threshold \(\theta_l\) is determined in training. For faster inference, Robust Norm requires empirical percentile which RNL doesn’t need, as \(p_l\) is trainable. Certainly, just using ANN loss will not guide the model to reduce the inference delay. This also requires additional loss design, which will be shown later.
2.3 Optimization for Fast Inference

Rate-based SNN models take rate coding as input, and the time average of the output spikes as output. In the conversion methods, the coding scheme mainly consists of two ways. One is Poisson coding, of which spikes obey the Poisson process. The other is constant coding, to cooperate with the reset-by-subtraction neuron. Constant coding is not a new thing, it can be regarded as an integrating ADC in the signal processing [Eng Jr and Matson, 1994]. The two primary rate coding forms take time for the firing rate to approach its expectations. For the constant coding, its accumulated current resembles the chi-square test in expectations. For the constant coding, its accumulated current coding forms take time for the firing rate to approach its expectations.

For layer $l$ in an ANN and the converted SNN with constant coding, given the simulated firing rate $\hat{r}_l$ and the real firing rate $r_l$, we have:

$$K_l < \frac{2\Omega_l}{t},$$

where $K_l$ denotes the abbreviation of $K(\hat{r}_l, r_l(t))$ in layer $l$, $\Omega_l = \|\hat{r}_l\|_1 \cdot \|\cdot\|_p$ denotes $L^p$ norm.

The detailed proof of Theorem 2 is described in the Appendix. $\Omega_l$ is named as Rate Inference Loss (RIL) of the $l$-th layer. Eq 8 indicates that $K_l$ is less than an inverse proportional curve to $t$.

For the convergence of the last layer (the $L$-th layer), if $\Omega_L$ reduces, $K_L$ will converge faster to 0 due to the Squeeze Theorem. That is, the real firing rate of the SNN approaches the simulated value of the ANN more faster, leading to faster and more stable outputs of SNN. However, considering that the network actually has inference delays layer by layer, a better solution is to reduce the average value of $\Omega_l (l = 1, 2, \ldots, L)$. Thus, the overall training loss is composed of the loss related to the task and RIL multiplied by hyperparameter $\lambda$.

$$\mathcal{L}'(f(x), y) = \mathcal{L}(f(x), y) + \lambda \sum_{i}^{L} \frac{\Omega_i}{L},$$

where $(x, y)$ is data tuple for training and $f(\cdot)$ is the network with L Rate Norm Layers. Based on Eq. 6, we are able to calculate the partial derivative of $\Omega_L$ w.r.t. $p_L$, and obtain:

$$\frac{\partial \Omega_L}{\partial p_L} = \sum_{i} \frac{\partial \Omega_L}{\partial \hat{r}_{L,i}} \frac{\partial \hat{r}_{L,i}}{\partial p_L}$$

$$= \sum_{i} \left[ \left( \frac{\hat{r}_{L,i}^2}{\|\hat{r}_{L,i}\|_2^2} - \frac{2 \hat{r}_{L,i} \|\hat{r}_{L,i}\|_1}{\|\hat{r}_{L,i}\|_2} \right) \left( -\frac{\hat{r}_{L,i}}{p_L} \right) \right]$$

$$= \frac{\|\hat{r}_{L,i}\|_1}{p_L \|\hat{r}_{L,i}\|_2^2},$$

Note that the design of K resembles the chi-square test in hypothesis testing. $\hat{r}$ and $r(t)$ denote the firing rates of all neurons in a certain layer. $\|\cdot\|_2$ indicates $L^2$ norm. The denominator $\|\hat{r}\|_2^2$ makes $K(\hat{r}, r(t))$ have scale invariance. Therefore, we can compare the fitting of the firing rate between different layers. Ideally, given enough “inference time”, $K(\hat{r}, r(t))$ will converge to 0. We believe K curve is an adequate metric as the neuron population encoding information is considered rather than any single neuron.
where \( \hat{r}_{L,i} \) denotes the \( i \)-th element of \( \hat{r}_L \).

Eq. 10 implies that the partial derivative of \( \Omega_L \) w.r.t. \( p_l \) is positive. Simply minimizing \( \Omega_L \) will reduce the neurons’ \( p_l \). The upper limit of \( p_l \) is unnecessary in this sense. Nevertheless, this will lead more neurons to saturation state and lose model accuracy. Thus, we jointly optimize the two losses and tune the hyper-parameter \( \lambda \) to reach the optimal trade-off between model accuracy and conversion loss.

So far, the current theories and analyses can also systematically explain the findings of Han et al. That is, reasonable scaling of the threshold can speed up the inference. If the threshold is set smaller, the Rate Inference Loss will decrease. Since the curve value and time are inversely proportional, it is equivalent to accelerating the decrease of the curve. On the contrary, if the threshold is too large, many neurons in the SNN will have low firing rates to produce accurate output. Of course, the threshold should not be too small to prevent more neurons from saturating and losing information. Fig. 4(b) shows the K curve after adjusting the threshold using the algorithm of Han et al.

### 2.4 Training for Accurate and Fast SNN

In Section 2.2, the \( p_l \) of the Rate Norm Layer is restricted to \([0, 1]\). This means if the Rate Norm Layer is directly trained, the simulated firing rate after clipping and scaling will inevitably bear information loss. The information loss here indicates that the output distribution due to cropping is different from the original distribution. This will make it difficult to take advantage of ANN’s performance. So the training strategy needs to be carefully designed. When \( p_l = 1 \) is fixed, \( z_l \) is clipped with \( \text{running\_max}(z_l) \), which has less information loss. After the network synaptic parameters are fixed, the neuron threshold starts training. The training goal at this stage is to reduce information loss and to reduce Rate Inference Loss. As the output \( f(x) \) of an ANN is \( \hat{r}_L \), the goal is to optimize (according to Eq. 9):

\[
\mathcal{L}'(\hat{r}_L; y) = \mathcal{L}(\hat{r}_L; y) + \lambda \frac{\sum_{L} \Omega_L}{L}, \quad (11)
\]

we decompose this goal into two stage goals (shown in Fig. 5):

Stage 1 is accuracy training, when network outputs \( \hat{r}_L \).

![Diagram of SNN training](image)

Figure 5: Diagram of SNN training. The training consists of two stages. It first trains for accuracy by adjusting weights. Then it optimizes for fast inference by adjusting firing thresholds.

| Network  | SNN Acc (%) | Conversion Loss (%) |
|----------|-------------|---------------------|
| MNIST    |             |                     |
| [Dahl et al., 2015] | Spiking NN | 98.6 | - |
| [Rueckauer et al., 2017] | - | 99.44 | 0.00 |
| This work (RNL) | 7-Layered CNN | 96.51 | 0.00 |
| This work (RNL) | AlexNet | 99.46 | -0.04 |
| CIFAR-10 |             |                     |
| [Sengupta et al., 2019] | ResNet-20 | 87.46 | +1.64 |
| [Sengupta et al., 2019] | VGG-16 | 91.55 | +0.15 |
| [Hunsberger and Eliasmith, 2015] | - | 83.54 | +0.18 |
| [Cao et al., 2015] | 7-Layered CNN* | 77.43 | +1.69 |
| [Han et al., 2020] | ResNet-20 | 91.36 | +0.11 |
| [Han et al., 2020] | VGG-16 | 93.63 | 0.00 |
| [Rueckauer et al., 2017] | - | 88.82 | +0.05 |
| This work (RNL) | VGG-16 | 92.86 | -0.04 |
| This work (RNL) | PreActResNet-18 | 93.45 | -0.39 |
| CIFAR-100 |             |                     |
| [Han et al., 2020] | VGG-16 | 79.33 | +0.29 |
| [Han et al., 2020] | ResNet-20 | 67.82 | +0.90 |
| This work (RNL) | VGG-16 | 75.02 | +0.54 |
| This work (RNL) | PreActResNet-18 | 75.10 | -0.45 |
| This work (RNL) | PreActResNet-34 | 72.91 | +0.80 |

Table 1: Best Accuracy Performance comparing with related methods. Values in the table represent the best accuracy and the accuracy loss of conversion (\( \text{Acc}_{\text{SNN}} – \text{Acc}_{\text{ANN}} \)). The structure of 7-Layered CNN is 32C3-P2-32C3-P2-32C3-P2-32FC10, which is different from the one with asterisk in the table.

The target is:

\[
\min_{W,b} \mathcal{L}^*(\hat{r}_L; y). \quad (12)
\]

Stage 2 is for fast inference. Assume the output of this stage is \( \hat{r}'_L \). Then the target is:

\[
\min_{\theta} \mathcal{T}(\hat{r}'_j, \hat{r}'_j^*) = 1 - \text{Cos}(\hat{r}'_j, \hat{r}'_j^*) + \lambda \sum_{L} \Omega_L. \quad (13)
\]

The cos distance is used to maintain neuron information. The detailed training and converting algorithm is described in Algorithm 1 in the Appendix.

### 3 Experiments

#### 3.1 Experiment Implementation

We validate our methods on the image recognition benchmarks, namely the MNIST\(^1\), CIFAR-10, CIFAR-100\(^2\) datasets. For MNIST, we consider a 7-layered CNN and AlexNet. For CIFAR-10, we use VGG-16 and PreActResNet-18 network structures. It is worth noting that we did not use the common ResNet as we think PreActResNet will help the training of Rate Norm Layers [He et al., 2016b]. For CIFAR-100, VGG-16, PreActResNet-18 and PreActResNet-34 are used. We present the simulation results and analysis in the following subsections.

#### 3.2 Accuracy Performance

We first evaluate the effectiveness of the proposed Rate Norm Layer. A network with Rate Norm Layers is trainable with backpropagation. When testing SNN performance, all Rate

1. [http://yann.lecun.com/exdb/mnist/](http://yann.lecun.com/exdb/mnist/)
2. [https://www.cs.toronto.edu/~kriz/cifar.html](https://www.cs.toronto.edu/~kriz/cifar.html)
Norm Layers are converted to I&F neuron models with \( \theta_l \) as the threshold. Table 1 shows the best accuracy of the converted SNNs compared with typical works. The converted SNNs achieve the state-of-the-art performance on MNIST and CIFAR-100 datasets, and reach a similar performance on CIFAR-10. For VGG-16 trained by CIFAR-100, the proposed method reaches top-1 accuracy 75.02\%, whereas the state-of-the-art ANN-SNN algorithm reaches 70.93\%. Conversion loss is considered here to evaluate the quality of the ANN-SNN conversion. As illustrated in Table 1, ANN-SNN conversion with Rate Norm Layers has low conversion loss, or even negative conversion loss, indicating that the converted SNN may outperform the original ANN. In contrast, the loss is usually positive for other methods, meaning that the performance of the converted SNN is not as good as ANN.

### 3.3 Fast Inference Performance

We test whether the proposed Rate Inference Loss can speed up inference, that is, speed up the convergence of the K curve. Fig. 6(a) and (b) show how the K curve and accuracy change over latency, where the dotted line in Fig. 6(a) represents the Max Norm method. As can be seen from Fig. 6(a), the K curves of the proposed method converge to 0 quickly, which are much faster than those of the Max Norm method. Thus the proposed method can implement fast inference. The inference performance can be observed in Fig. 6(b). The proposed method reaches an accuracy of 85.40\% using 32 time-steps, whereas the methods of Max Norm, Robust Norm, and RMP-SNN reach 10.00\%, 43.03\% and 63.30\% at the end of 32 time-steps. Moreover, the proposed method achieves an accuracy above 90\% using only 52 time-steps, which is 8.6 times faster than Max Norm that uses 446 time-steps. Detailed accuracy comparison on time T is shown in Table 2.

![Image](image-url)

Figure 6: Fast inference performance of SNNs converted from VGG-16 trained on CIFAR-10. (a) The K curve over time. For readability, 4 of the 16 layers of VGG-16 are extracted to display the K curves. (b) Accuracy curves over time. The dotted line only indicates the tendency due to the lack of data. (c) \( \Omega \) and threshold distribution.

### 3.4 Energy Estimation of Neuromorphic Hardware

SNNs have a considerable potential on neuromorphic chips. One of the benefits is to reduce the energy budget. To study the energy efficiency of fast reasoning, we use the energy model proposed by Cao et al. [Cao et al., 2015] to model the energy consumption on neuromorphic chips. Assume that a spike activity would bring about the energy consumption of \( \alpha \) Joules and 1 time-step takes 1ms. Then the power model is defined as:

\[
P = \frac{total \ spikes}{3600} \times \alpha (\text{Watts})
\]

The previous energy analysis mainly focused on total energy consumption during inference. However, in real application scenarios, the total energy consumed before the model reaches reliable accuracy is more important. In this regard, we evaluate the performance of the proposed method and the Max Norm method on energy. Fig. 7(a) is the power histogram over time. Due to the lower threshold, the power of our model is relatively high. The integral of the deep color area represents the energy consumed to achieve 90\% accuracy. The energy consumption of the proposed model is only 0.265 times of the Max Norm method when it reaches 90\% accuracy. This means that the 8.6 \times reasoning speedup will not bring much more energy consumption. Fig. 7(b) shows the logarithmic ratio of energy and inference speedup. Our model exhibits the properties of “fast reasoning” and “energy efficiency”.

### 4 Conclusions

This paper proposes a method to convert conventional ANNs to SNNs. The Rate Norm Layer is introduced to replace ReLU for optimal conversion. Besides, we quantify the fit between the ANN activation and the firing rate of the converted SNN by an optimal fit curve. The inference time can
be reduced by optimizing the coefficient of the upper bound of the fit curve, namely Rate Inference Loss. Thus, a two-staged learning scheme is proposed to obtain fast and accurate deep SNNs. Experimental results demonstrate that our methods achieve low accuracy loss and fast reasoning with deep structures such as VGG and PreActResNet.

Acknowledgment
This work was supported by the National Natural Science Foundation of China (62027804, 61825101, 62088102 and 61961130392).

Appendix A: Proofs
Here we provide the theoretical proofs of theorems and lemmas in this paper.

**Lemma 1.** For a spiking neural network consisting of the reset-by-subtraction neurons mentioned in Eq. 3, assume that \( W_{l-1} \) and \( b_{l-1} \) are the parameters for layer \( l-1 \). Then when \( t \to \infty \), the relation of the firing rate \( r_l(t) \) and \( r_{l-1}(t) \) is given by:

\[
r_l = \text{clip} \left( \frac{W_{l-1}r_{l-1} + b_{l-1}}{v_{th,l}}, 0, 1 \right),
\]

where \( \text{clip}(x, 0, 1) = x \) when \( x \in [0, 1] \), \( \text{clip}(x, 0, 1) = 1 \) when \( x > 1 \), and \( \text{clip}(x, 0, 1) = 0 \) when \( x < 0 \).

**Proof.** For the reset-by-subtraction spiking neurons formulated by Eq. 3 in the main text, we can stack the equations and get the discrete function between spikes of layer \( l \) and layer \( l-1 \):

\[
v_l(t) - v_l(t-1) = W_{l-1}s_{l-1}(t) + b_{l-1} - v_{th,l} s_l(t).
\]

By summing the left and right expressions over time and dividing \( t_{th,l} \) on both sides, the equation can be reformulated as:

\[
r_l(t) = \frac{\sum s_l(t)}{t} = \frac{W_{l-1}r_{l-1} + b_{l-1} - v_l(t)}{t_{th,l}},
\]

where \( r_l(t) \) denotes the firing rates of all neurons in layer \( l \).

From the membrane potential updating function (Eq. 3), \( v_l(t) \) is in the range of \([0, v_{th,l}]\), thus we have:

\[
\lim_{t \to \infty} \frac{v_l(t)}{t_{th,l}} = 0.
\]

As the value of \( s_l(t) \) can only be 0 or 1, the firing rate \( r_l(t) \) is strictly restricted in \([0, 1]\). When \( t \to \infty \), it’s straightforward to conclude that

\[
r_l = \text{clip} \left( \frac{W_{l-1}r_{l-1} + b_{l-1}}{v_{th,l}}, 0, 1 \right).
\]

**Theorem 1.** For an \( L \)-layered ANN with the ReLU activation and an \( L \)-layered SNN with the reset-by-subtraction neurons, assume that \( W_{ANN}^{l-1}, b_{ANN}^{l-1} \) are the parameters for layer \( l-1 \) of the ANN, and \( W_{SNN}^{l-1}, b_{SNN}^{l-1} \) are the parameters for layer \( l-1 \) of the SNN. \( \max_{l} \) is the maximum activation of layer \( l \) in ANN, and \( v_{th,l} \) is the firing threshold of layer \( l \) in SNN when \( t \to \infty \). The ANN can be converted to the SNN (Eq. 1 equals Eq. 4) if for \( l = 1, 2, \ldots, L \), the following equations hold:

\[
\frac{W_{ANN}^{l-1}}{v_{th,l}} s_l = \frac{W_{SNN}^{l-1} r_{l-1} + b_{SNN}^{l-1}}{v_{th,l}} r_{l-1},
\]

\[
\frac{b_{ANN}^{l-1}}{v_{th,l}} = \frac{b_{SNN}^{l-1}}{\max_{l}}.
\]

**Proof.** The conversion of ANN to SNN is built on the basis of the equivalent of SNN firing rate and ANN activation. Considering spiking neurons of layer \( l \) and \( l-1 \) in an SNN, the relationship between the firing rate \( r_l(t) \) and \( r_{l-1}(t) \) is:

\[
r_l = \text{clip} \left( \frac{W_{SNN}^{l-1} r_{l-1} + b_{SNN}^{l-1}}{v_{th,l}}, 0, 1 \right).
\]

Note that the firing rate \( r_i(i = 1, 2, \ldots, L) \) in Eq. 7 is restricted in \([0, 1]\). But the ANN activation (ReLU) only satisfy \( a_i \geq 0 \) (Eq. 1 in the main text). In fact, for countable limited dataset, the activation generated by the network is also upper bounded. Assume that the upper bound for the output of all ReLU-based artificial neurons in layer \( i \) is \( \max_{i} \), we have:

\[
0 \leq a_i \leq \max_{i}.
\]

Let \( z_i = \frac{a_i}{\max_{i}} \), then \( 0 \leq z_i \leq 1 \). According to Eq. 1, the activation of layer \( l \) and \( l-1 \) satisfy:

\[
a_l = \max_{i} \left( W_{ANN}^{l-1} a_{i-1} + b_{ANN}^{l-1}, 0 \right)
\]

As the \( a_l \) clipped by \( \max_{i} \) equals the original \( a_l \). Then:

\[
a_l = \text{clip} \left( W_{ANN}^{l-1} a_{i-1} + b_{ANN}^{l-1}, 0, \max_{i} \right)
\]

By dividing \( \max_{i} \) on both sides of Eq. 10 and substituting \( a_i \) by \( z_i \max_{i} \), we have:

\[
z_l = \text{clip} \left( W_{ANN}^{l-1} z_{l-1} \max_{l-1} + b_{ANN}^{l-1}, 0, \max_{l} \right)
\]

Comparing Eq. 7 and Eq. 11. We can conclude that Eq. 7 equals Eq. 11 if for \( l = 1, 2, \ldots, L \), the following equations hold:

\[
\frac{W_{ANN}^{l-1}}{v_{th,l}} = W_{SNN}^{l-1} \frac{\max_{l}}{\max_{i}}, \quad \frac{b_{ANN}^{l-1}}{v_{th,l}} = \frac{b_{SNN}^{l-1}}{\max_{l}}.
\]
Theorem 2. For layer $l$ in an ANN and the converted SNN with constant coding, given the simulated firing rate $\hat{r}_l$ and the real firing rate $r_l$, we have:

$$K_l < \frac{2\Omega_l}{t},$$

where $K_l$ denotes the abbreviation of $K(\hat{r}_l, r_l(t))$ in layer $l$, $\Omega_l = \|r_l\|_p \cdot : \| \cdot \|_p$ denotes $L^p$ norm.

Proof. Consider that firing is a cumulative and rounded firing process. When $\hat{r}_l$ is given, $r_l(t)$ is approximate as $\frac{\hat{r}_l(t)}{t}$, also $\hat{r}_{l,i} - \frac{1}{t} < \frac{\hat{r}_{l,i}(t)}{t} < \hat{r}_{l,i}$, where $\hat{r}_{l,i}$ denotes the $i$-th element of the vector $\hat{r}_l$. For the $i$-th layer, we have:

$$\left\| r_l(t) - \hat{r}_l \right\|_2 = \left\| \frac{\hat{r}_l}{t} - \hat{r}_l \right\|_2 = \left\| \sum_i \left[ \frac{\hat{r}_{l,i}(t)}{t} - \hat{r}_{l,i} \right] \right\|_2$$

$$= \sum_i \left( \frac{\hat{r}_{l,i}(t)}{t} - \hat{r}_{l,i} \right)^2 - 2\hat{r}_{l,i} \frac{|\hat{r}_{l,i}(t)|}{t} + \hat{r}_{l,i}^2$$

$$< \sum_i \left[ \hat{r}_{l,i}^2 - 2\hat{r}_{l,i} \left( \hat{r}_{l,i} - \frac{1}{t} \right) + \hat{r}_{l,i}^2 \right]$$

$$= \sum_i \left[ \hat{r}_{l,i}^2 - 2\hat{r}_{l,i}^2 + 2\hat{r}_{l,i} \frac{1}{t} + \hat{r}_{l,i}^2 \right]$$

$$= \sum_i \frac{2\hat{r}_{l,i}}{t} = 2\left\| \frac{\hat{r}_l}{t} \right\|_1$$

The last equality holds as simulated firing rate $\hat{r}_l \geq 0$. Thus the sum of all items in $\hat{r}_l$ equals its $L^1$ norm. Now we conclude that:

$$K_l = \frac{\left\| r_l(t) - \hat{r}_l \right\|_2^2}{\left\| \hat{r}_l \right\|_2^2} < \frac{2\left\| \hat{r}_l \right\|_1}{\left\| \hat{r}_l \right\|_2^2} = \frac{2\Omega_l}{t}$$

$\square$

Appendix B: Supplementary of Methods and Experiments

The training and converting algorithm is described in Algorithm 1. In Stage 2, since each trainable $p_i$ will scale the output, to reduce the instability of threshold training on the deep model, all layers share the same $p_i$ when training. Besides, to limit $p_i \in [0, 1]$ of Rate Norm Layer, use $\text{sigmoid}(p'_i)$ in place of $p_i$.

The experiments are conducted on the PyTorch platform. The GPU used in training is NVIDIA GeForce RTX 2080 Ti.

Algorithm 1 Mini-batch training and converting a spiking neural network from a source ANN with Rate Norm Layers. ‘r_max’ is short for ‘running_max’.

Require: A spiking neural network $f_{SNN}$ with thresholds $\{v_{th,k}|k=1,2,\ldots,L\}$; A network with $L$ Rate Norm Layers $f_{ANN}$; Dataset $D$ for training; Number of epochs for Stage 1 $epoch1$; Number of epochs for Stage 2 $epoch2$

Ensure: $p_k=1.0, r_{max,k}=1.0$ $(k = 1, 2, \ldots, L)$; momentum parameter $m = 0.1, \lambda = 0.5$

1: {Stage 1: Accuracy Training for $f_{ANN}$}
2: for $e = 1$ to $epoch1$ do
3: for length of Dataset $D$ do
4: Sample minibatch $(x, y)$ from $D$
5: $r_0^x = x$
6: for $k = 1$ to $L$ do
7: $r_{max,k} = (1-m) r_{max,k} + m \max(x_{k-1})$
8: $\theta_k = p_k r_{max,k}$
9: $r^x_k = \text{clip}([W_{k-1} r^x_{k-1} + b_{k-1}] / \theta_k, 0.1]$ end for
11: Loss = $\mathcal{L}(r^x_k; y)$
12: Backward propagation through network
13: Update $W_k$ and $b_k (k = 1, 2, \ldots, L - 1)$
14: end for
16: {Stage 2: Fast Inference Training for $f_{SNN}$}
17: for $e = 1$ to $epoch2$ do
18: for length of Dataset $D$ do
19: Sample minibatch $(x, y)$ from $D$
20: $r_0^x = x$
21: for $k = 1$ to $L$ do
22: $r_{max,k} = (1-m) r_{max,k} + m \max(x_{k-1})$
23: $\theta_k = p_k r_{max,k}$
24: $r^x_k = \text{clip}([W_{k-1} r^x_{k-1} + b_{k-1}] / \theta_k, 0.1]$ end for
26: for $k = 1$ to $L$ do
27: $r_{max,k} = (1-m) r_{max,k} + m \max(x_{k-1})$
28: $\theta_k = p_k r_{max,k}$
29: $r^x_k = \text{clip}([W_{k-1} r^x_{k-1} + b_{k-1}] / \theta_k, 0.1]$ end for
31: Loss = $\mathcal{J}(\tilde{r}^x_k, \tilde{r}^x_k)$ (Eq.13)
32: Backward propagation through network
33: Update $p_k (k = 1, 2, \ldots, L)$
34: end for
36: {Converting SNN $f_{SNN}$ from pre-trained $f_{ANN}$}
37: for $k = 1$ to $L$ do
38: $f_{SNN}.W_k = f_{ANN}.W_k$
39: $f_{SNN}.b_k = f_{ANN}.b_k$
40: $f_{SNN}.v_{th,k} = f_{ANN}.v_k$
41: end for
42: return $f_{SNN}$
