TDSNN: From Deep Neural Networks to Deep Spike Neural Networks with Temporal-Coding

Lei Zhang,[1,2,3] **Shengyuan Zhou,**[1,2,3] **Tian Zhi,**[1,3] **Zidong Du,**[1,3] **Yunji Chen**[*1,2]
[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]Cambricon Tech. Ltd
{zhanglei03, zhousy, zhitian, duzidong, cyj}@ict.ac.cn

Abstract

Continuous-valued deep convolutional networks (DNNs) can be converted into accurate rate-coding based spike neural networks (SNNs). However, the substantial computational and energy costs, which is caused by multiple spikes, limit their use in mobile and embedded applications. And recent works have shown that the newly emerged temporal-coding based SNNs converted from DNNs can reduce the computational load effectively. In this paper, we propose a novel method to convert DNNs to temporal-coding SNNs, called *TDSNN*. Combined with the characteristic of the leaky integrate-and-fire (LIF) neural model, we put forward a new coding principle *Reverse Coding* and design a novel *Ticking Neuron* mechanism. According to our evaluation, our proposed method achieves 42% total operations reduction on average in large networks comparing with DNNs with no more than 0.5% accuracy loss. The evaluation shows that TDSNN may prove to be one of the key enablers to make the adoption of SNNs widespread.

Introduction

SNNs are considered to be the third generation of neural networks which are more powerful on processing of both spatial and temporal information (Maass 1997). However, it is difficult to train an SNN directly. A series of works have tuned the backpropagation algorithms to fit for the SNN models (Bohte, Kok, and Han 2000; Lee, Delbruck, and Pfeiffer 2016). Although they achieve satisfactory results in simple tasks like MNIST (Lecun et al. 1998), the training methods can hardly be scaled into deep SNN models to solve more complex tasks like ImageNet (Russakovsky et al. 2014). Meanwhile, the researches on the biological training methods also meet the same problem. These works have put many existing brain mechanisms into the modeling of SNN including spike timing dependent plasticity (STDP) and long-term potentiation or depression, short-term facilitation or depression, hetero-synaptic plasticity, etc. Recently, Zhang et al. (Zhang et al. 2018) proposed a novel multilayer SNN model which applies biological mechanisms and achieves 98.52% on MNIST dataset, but its performance is unknown when applied on larger datasets.

[*]Yunji Chen (cyj@ict.ac.cn) is the corresponding author.

Unlike SNNs, deep neural networks (DNNs) have been able to perform the state-of-the-art results on many complex tasks such as image recognition (Krizhevsky, Sutskever, and Hinton 2012; Krizhevsky 2009; Simonyan and Zisserman 2014; He et al. 2015), speech recognition (Abdel-Hamid et al. 2012; Sainath et al. 2013; Hinton et al. 2012), natural language processing (Kim 2014; Severyn and Moschitti 2015) and so on. But heavy computation load promotes researchers to find more efficient approach to deploy them in mobiles or embedded systems. This inspires the SNN researchers that a fully-trained DNN might be slightly tuned to be directly converted to a SNN without complicated training procedure. Beginning with the work of (Perezcarrasco et al. 2013), where DNN units were translated into biologically inspired spiking units with leaks and refractory periods, continuous efforts have been made to realize this idea. After a series of success in transferring deep networks like Lenet and VGG-16 (Cao, Chen, and Khosla 2015; Diehl et al. 2015; Rueckauer et al. 2017), now the rate-coding based SNN can achieve state-of-the-art performance with minor accuracy loss even in the conversion of complicated layers like Max-Pool, BatchNorm and SoftMax.

However, weak points in rate-coding based SNN are obvious. Firstly, the rate-coding based SNN could become more accurate with the increasement of the simulation duration and the average firing rate of its neurons. But it may lose potential performance advantage over the DNNs as the firing rates increase (Rueckauer and Liu 2018). Secondly, part of the accuracy loss in the conversion occurs at the parameter determination procedure. In rate-coding based SNN, determination of important parameters like firing threshold seriously affect the final accuracy while non deterministic methods have been proposed to eliminate the loss.

This poses challenges to researchers to find conversion methods based on another coding scheme—temporal coding. Neurons based on temporal-coding make full use of the spike time to complete the transmission of information, and the number of spikes is significantly reduced. Also, temporal coding has been proved to be efficient in computing even in biological brains (Van and Thorpe 2001). The temporal-coding based neurons apply a so-called time-to-first-spike (TTFS) scheme and each neuron fires at most once during the forward inference (Thorpe, Delorme, and Rullen 2001). Obviously, temporal-coding based SNN is more computationally efficient than rate-coding based SNN. Recently Rueckauer and Liu (Rueckauer and Liu 2018) proposed a conversion method using sparse temporal coding, but it is only suitable for shallow neural network models.

In this paper, we put forward a novel conversion method called *TDSNN*, transferring DNNs to temporal-coding SNNs with only trivial accuracy loss. We also propose a new encoding scheme *Reverse Coding* along with a novel *Ticking Neuron* mechanism. We address three challenges during the process of converting as follows. Firstly, we maintain the coding consistency in the entire network after mapping. Secondly, we eliminate the conversion error from DNNs to SNNs. Thirdly, we maintain accuracy in SNN with spike-once neurons. The details will be discussed later in the *Reverse Coding* part. According to our evaluation, we achieve 42% total operations reduction on average in large networks comparing with DNNs with no more than 0.5% accuracy loss. The experimental results show that our method is an efficient and high-performance conversion method for temporal-coding based SNN.

Background

In this section, we will introduce two common neural models. One is integrate-and-fire(IF) model, the other is leaky integrate-and-fire (LIF) model.

Integrate-and-fire (IF) model

We introduce integrate-and-fire(IF) model at first, which is widely used in previous rate-coding based conversion approaches. In IF models, each neuron will accumulate potential from input current and fire a spike when its potential reaches the threshold. Although successful conversions have been made from DNN neurons with ReLU activations to IF neurons, the process of the transformation still exists unsatisfactory problems. For example, it is difficult to accurately determine thresholds in these conversion methods. Although Diehl et al. (Diehl et al. 2015) has proposed data-based and model-based threshold determination methods, the converted SNN still suffers unstable accuracy loss comparing with the re-trained DNN model. What's worse, IF neurons implement no time-dependent memory. Once a neuron receives a below-threshold action potential at some time, it will keep retaining that voltage until it fires. Obviously, it is not in line with the neural behavior in neural science.

Leaky Integrate-and-fire (LIF) model

To better model neural behavior in the spike-threshold model, researchers in neural science field proposed a simplified model—Leaky Integrate-and-fire (LIF) model (Koch and Segev 1998), which is derived from the famous Hodgkin–Huxley model (Hodgkin and Huxley 1990). Generally, it is defined as following:

$$I(t) - \frac{V(t)}{R} = C \cdot \frac{dV(t)}{dt}, \tag{1}$$

where R is a leaky constant, C is the membrane capacitance, $V(t)$ is the membrane potential and $I(t)$ is considered as the input stimulus.

Considering that the inputs are instantaneous currents that are generated from discrete-time neural spikes in an SNN model, the equation (1) turns to:

$$\frac{V(t)}{L} + \frac{dV(t)}{dt} = \sum_i \omega_i \cdot I_i(t), \tag{2}$$

where ω_i denote the synapse strength of the connections in SNN, L is the leak-related constant. If there exists no continuous spikes, the neuron potential will decay as time goes by. The LIF model processes the time information by adding a leaky term, reflecting the diffusion of ions that occurs through the membrane when some equilibrium is not reached in the cell. If no spike occurs during the time interval from t_1 to t_2, the potential of the LIF neuron will change as following:

$$V(t_2) = V(t_1) \cdot exp(-\frac{t_2 - t_1}{L}). \tag{3}$$

And if no no spike occurs in this neuron, the final potential accumulation P at time T_{il} will be:

$$P(T_{il}) = \sum_i \omega_i \cdot exp(-\frac{T_{il} - t_i}{L}). \tag{4}$$

This implies that the pre-synaptic neuron's potential contribution to post-synaptic neurons is positively correlated with the firing time of pre-synaptic spike. This nonlinear characteristic has never been put good use in previous conversion methods. We make full use of these features to build a conversion method for temporal-coding SNN.

Theoretical Analysis

In this section, we present details of our conversion method. We first introduce the *Reverse Coding* guideline and *ticking neuron* mechanism, respectively. Then, we analyze theoretically on all the requirements of conversion method, including temporal-coding function and important parameters.

Reverse Coding

There has been massive number of influential works in the practice of encoding input into a single spike before our work. For example, Van et al. (Van and Thorpe 2001) proposed a rank order based encoding scheme, in which the inputs are encoded into specific spiking order. The potential contribution of later-spike neurons will be punished with a delay factor so that the ones fire earlier will contribute a major part of potential accumulation to post-synaptic neurons. Combined with LIF neurons, this coding method has been proven to achieve a Gaussian-difference-filtering effect, which is helpful for extracting features. But these guidelines failed to serve well when considering the conversion of DNN to SNN.

Unlike these works in which significant inputs are encoded into earlier spike times, we follow an opposite coding guideline based on characteristics of LIF neurons and we name it *Reverse Coding*:

The stronger the input stimulus is, the later the corresponding neuron fires a spike.

This is consistent with the characteristics of leaky-IF neurons, where post-synaptic neurons will assert most impressive contribution to the recent-spike neurons.

Determining the principle of coding schemes moves one step closer to build a conversion method for SNN with temporal-coding. Following *Reverse Coding*, we still need to tackle the following challenges to build a conversion method for SNN with temporal-coding:

- **How to maintain coding consistency in the entire network after mapping.** Even if the input of the first layer is encoded according to *Reverse Coding*, there is no guarantee that the neurons in the subsequent layers will also spike exactly in the same way.
- **How to eliminate the conversion error from DNNs to SNNs.** A bunch of parameters(threshold, spike frequency, presentation time etc.) need to be determined in previous rate-coding based conversion methods and it still lacks strict theoretical evidence to minimize the error caused by parameter selection.
- **How to maintain accuracy in SNN with spike-once neurons.** This requires the converted SNN makes full use of the nonlinear characteristics of neuron model and also the adjustments in DNN should be well designed.

Next, we will tackle these challenges by introducing the *Ticking Neuron* mechanism below.

Ticking Neuron Mechanism

Converting the weights of DNN into that of SNN directly is the most straightforward way, which also occurred in previous rate-coding based conversion methods. However, since none inhibitory mechanisms are applied, the post-synaptic neurons would spike at any time once their potentials exceed the threshold. In such case, it would be difficult to control the spiking moment and spike times along with the appropriate threshold. What's worse, those neurons in SNN, whose corresponding neuron output in DNN is large, may issue spikes at an earlier time. It violates the *Reverse Coding* principle we introduced above.

To address the challenges above, we propose an auxiliary neuron called *ticking neuron* and a corresponding spike processing mechanism. As it shows in figure 1, the post-synaptic neurons will be inhibited to fire once the process begins, reflecting a common refractory period in SNN. The inhibition will last until time T_{il}(T_{il} must be large enough to cover all the pre-synaptic spikes, at least it could be the firing time of the neuron firing the last spike). During that time interval, each of the pre-synaptic neurons will only fire one spike following *Reverse Coding*. The *ticking neuron* will record the time T_{il} and update its connections' weight ω_{tick} to $f(T_{il})$ (which means ω_{tick} is only dependent on T_{il} and if T_{il} is a pre-determined constant it could also be pre-determined). After that, *ticking neuron* begins to issue spikes at each time step until each of the post-synaptic neurons has fired one spike. Those neurons has already fired one spike will be inhibited to fire(could be considered as being in a very long self-inhibitory period).

To make the LIF neuron whose corresponding neuron in DNN outputs a large value fire at a later time point, the original weights in DNN layers need to be negated. According to equation (4), for those LIF neurons whose corresponding output value in DNN is negative, they will spike immediately once the inhibition is released. While for those post-synaptic neurons whose corresponding output in DNN is positive, their potential will be at a negative point when all the pre-synaptic neurons fires. After that, the *ticking neuron* begins to fire, increasing their potential until reach the given threshold. Clearly, the larger the output in DNN, the later the corresponding neurons in SNN will fire a spike.

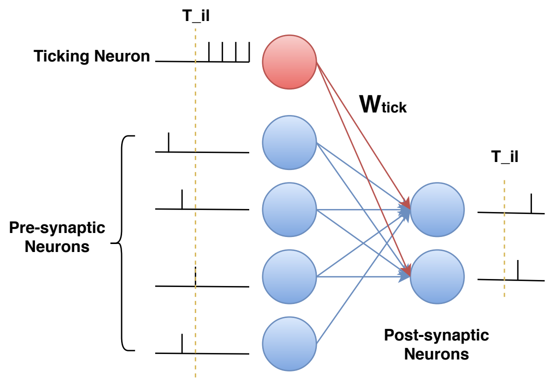

Figure 1: A Layer with a ticking neuron in SNN.

Why *ticking neuron* is necessary? Because it guarantees that all of the neurons can fire spikes. Supposing that if we remove the neuron, those post-synaptic neurons with a negative potential will slowly approach the reset potential. It will never fire a spike unless a threshold below reset potential is set for it, which is not consistent with existing SNN findings and neural science evidence.

Now all that is required is the strict calculations of spike timing to achieve a lossless conversion.

Mapping Synapse-based Layer

Synapse-based layers refer to those layers with connections of exact weight in the DNN, including convolutional layers (*CONV*), inner-product layers (*IP*), pooling layers (*POOL*), etc. These layers are the fundamental layers of DNN and also the most computationally intensive layers. We will show how to convert these layers to corresponding layers in SNN applied with mechanisms mentioned above.

Supposing that the firing threshold of post-synaptic neuron is θ, the last firing time of pre-synaptic neurons is T_{il} and the current below threshold potential is P. Clearly, the spike timing T_o (considering the output layer's inhibition is released at time 0) of the post-synaptic neuron is the minimum positive integer that obeys the following equation:

$$\sum_{t=0}^{T_o} \omega_{\text{tick}} \cdot \exp(-\frac{T_o - t}{L}) + P \cdot \exp(-\frac{T_o}{L}) \geq \theta, \tag{5}$$

where L is the leaky constant described before. Solving this equation, we obtain:

$$T_o = \lceil L \cdot \ln(\frac{(-P) \cdot (1 - \exp(-1/L)) + \omega_{\text{tick}}}{\omega_{\text{tick}} - \theta \cdot (1 - \exp(-1/L))}) \rceil. \tag{6}$$

In order to reduce the number of parameters to be determined, the above equation can be simplified by setting $\theta = 0$. Finally, the equation (6) turns out to be:

$$T_o = \lceil L \cdot \ln(\frac{(-P) \cdot (1 - \exp(-1/L))}{\omega_{\text{tick}}} + 1) \rceil. \quad (7)$$

Note that the output positive value of DNN neurons A will be mapped into a much smaller negative potential P due to the leaky effect of leaky-IF neurons, which is:

$$-P = A \cdot \exp(-\frac{T_{il}}{L}). \quad (8)$$

Combining equation (7) with equation (8), we will get the connection weight of ticking neuron is as following:

$$\omega_{\text{tick}} = (1 - \exp(-\frac{1}{L})) \cdot \exp(-\frac{T_{il}}{L}). \quad (9)$$

This is in line with the constraints we mentioned in the previous section that ω_{tick} is only dependent on T_{il}. If T_{il} is a predetermined large constant, ω_{tick} becomes a constant too. If T_{il} is set as the last firing time of pre-synaptic neurons, then ω_{tick} dynamically updated. Also, the temporal-coding function $T(x)$ will be obtained by combining equation (7)(8)(9):

$$T(x) = \lceil L \cdot \ln(x + 1) \rceil. \quad (10)$$

Correspondingly, a new activate function is needed to be deployed right before these synapse-based layers:

$$G(x) = \begin{cases} \exp(\frac{1}{L} \cdot \lceil L \cdot \ln(x+1) \rceil) & x \geq 0, \\ 1 & x < 0. \end{cases} \quad (11)$$

Note that negative input stimulus is avoided in our work just like other conversion methods did, as *negative neurons* are neither necessary nor in line with existing neural science.

Bias term was explicitly excluded in most of the rate-coding based conversion methods. Rueckauer et al. (Rueckauer et al. 2017) proposed the bias with an external spike input of constant rate proportional to the DNN bias. In temporal-coding SNN, determining the firing timing for bias neuron is easy. Here the bias neuron is considered as an input with a numerical value of 1, thus it will be encoded into a spike timing according to equation (10). The connection weights of bias neurons will also be negated after re-training of DNN.

Applied the mechanisms and coding functions above, the synapse-based layers in DNN can be accurately mapped to spike-time-based ones in SNN.

Mapping Max-Pool

Max-Pool layer is a commonly used down-sampling layer in DNN, replacing it with *Average-Pool* will inevitably decrease the DNN accuracy. But it is non-trivial to compute maxima with spiking neurons in SNN. In the rate-coding based SNN (Rueckauer et al. 2017), the authors proposed a simple mechanism for spiking max-pooling, in which output units contain gating functions that only let spikes from the maximally firing neuron pass, while discarding spikes from other neurons. Their methods prove to be efficient in rate-coding based SNN but can not be applied here in temporal-coding SNN.

Here we show how the lossless mapping of the *Max-Pool* layer can be done assuming that the weight of the ticking neuron ω_{tick} and the weights $-\omega_k$ in the SNN max-pool kernel (the weights in this layer are also negative). Supposing that the size of the pooling kernel is N (usually equals $KX \cdot KY$, which stands for the kernel width in x and y direction respectively), firing threshold of post-synaptic neurons is 0, and the weights of the mapped kernel are equally distributed.

To determine the weights of ω_{tick} and ω_k, two extreme scenarios should be considered. One is that post-synaptic neurons will accumulate a strong negative potential if all the pre-synaptic neurons in the kernel fire at time T_m ($T_m \leq T_{il}$). The other is that post-synaptic neurons will accumulate a weaker negative potential if there is only one pre-synaptic neurons fires at time T_m. It must be guaranteed that in both extreme cases, the spike timing T_o of a post-synaptic neuron equals T_m.

Taking the potential firing time of post-synaptic neurons in both cases into the equation (7), we obtain the equations 12 and derive the constraints shown as 13, where $C_L = (1 - \exp(-1/L))$ is a constant.

$$\begin{cases} L \cdot \ln(\frac{\omega_k \cdot (1 + (N-1) \cdot \exp(-\frac{T_m}{L}))}{\omega_{\text{tick}}/C_L} + 1) > T_m - 1 \\ L \cdot \ln(\frac{N \cdot \omega_k}{\omega_{\text{tick}}} \cdot C_L + 1) < T_m \end{cases} \quad (12)$$

$$\begin{cases} \frac{\omega_{\text{tick}}}{\omega_k} > \frac{C_L \cdot N}{\exp(\frac{T_m}{L}) - 1} \\ \frac{\omega_{\text{tick}}}{\omega_k} < \frac{C_L \cdot (1 + (N-1) \cdot \exp(-\frac{T_m}{L}))}{\exp(\frac{T_m - 1}{L}) - 1} \end{cases} \quad (13)$$

Notice that $\frac{\omega_{\text{tick}}}{\omega_k}$ is only dependent on T_m. If $T_m = 1$, only the left side of the above inequality is needed. Considering that the inequality holds if the left side is smaller than the right side, the constraints on L when $T_m > 1$ is obtained:

$$\frac{N}{\exp(T_m/L - 1)} < \frac{1 + (N-1) \cdot \exp(-T_m/L)}{\exp((T_m - 1)/L) - 1} \quad (14)$$

Because ω_k, L and N are predetermined, ω_{tick} is only dependent on the presentation time T_{il} of pre-synaptic neurons. As a result, the lossless mapping of the *Max-Pool* layer can be done by selecting parameters according to equations (13)(14).

However solving such constraints for various N and L would be complicated and the existence of solutions needs to be determined under many circumstances. Here we present a simplified solution for mapping *Max-Pool* layer under our mechanism. By setting the firing threshold $\theta = N$, $\omega_k = 1$ and canceling the use of ticking neuron and leaky effect, the firing moment of the output neuron is equal to the spiking moment of the latest firing neuron in the input kernel. In this way, *Max-Pool* is realized in SNN with *Reverse Coding*.

Network Conversion

In the section, we will introduce the specific conversion methods and the forward propagation procedure for the converted SNN based on the theoretical analysis.

DNN Adjustment and Re-training

As shown in figure 2, fetching weights from a fitted DNN model consists of following steps. Same as the previous rate-coding based conversion methods, a minor adjustment on the original DNN network is needed.

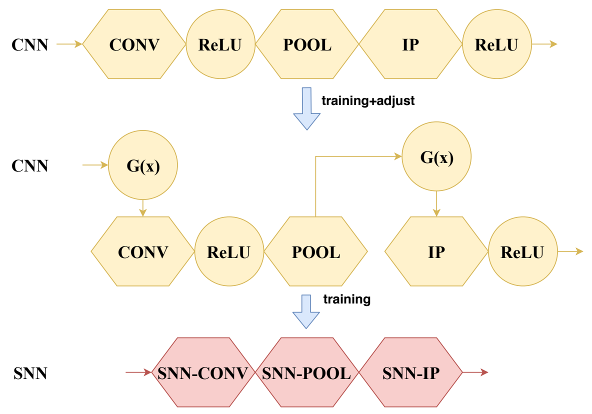

Figure 2: Conversion procedure.

Firstly, ReLU activations are deployed right after all the synapse-based layers, which is a classic structure in classic network models such as Alexnet, VGG-16. The adjusted network is trained through commonly used techniques for DNN training.

Secondly, the activation function $G(x)$ (see equation (11)) derived from the temporal-coding function is put right before the synapse-based layers. Retraining the network until the network's loss converges.

Finally, weights of synapse-based layers will be negated (except for the output layer, as it produces a float value for final decision). Those layers with no weights such as *Max-Pool* layer will be converted to a SNN layer with weighted connections. The determination of ω_k and ω_{tick} have already been provided in the last section.

Forward Propagation of SNN

The structure of each layer in the converted SNN is organized as Figure 1. According to the previous theoretical analysis, the entire computing procedure is quite different from previous temporal-coding or rate-coding SNNs. So we propose a new propagation algorithm of each layer in the converted SNN, described in Algorithm 1.

Each layer in the converted SNN will follow this algorithm to process spikes. The operation of the entire SNN network can be seen as that inhibition period and active period alternately appear in a pipeline, as shown in the figure 3. The active period of current layer is also the inhibitory period of the next layer. For the last layer of the network, we only accumulate the potential of the output neurons. The winner is the neurons with the largest potential, representing the final result.

```
Algorithm 1 Layer propagation in SNN
Require: pre-synaptic neurons that only spike once, inhibition time is $T_{il}$, a ticking neuron functions after $T_{il}$ with weights $\omega_{\text{tick}}$ connected to all the post-synaptic neurons.
Ensure: post-synaptic neurons only spike once.
1: Inhibitory Period. Post-synaptic neurons accumulate potential according to pre-synaptic spikes. All of them are inhibited to fire until $T_{il}$.
2: Determine Parameters. Update weights $\omega_{\text{tick}}$ of the ticking neuron according to $T_{il}$ and equation (9)(13).
3: Active Period. The ticking neuron fires at each time step, increasing potential of post-synaptic neurons. Then the post-synaptic neurons fire according to the LIF neural model.
```

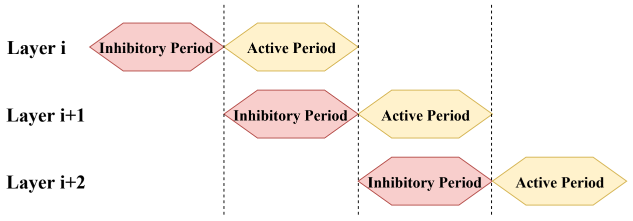

Figure 3: Propagation pipeline.

Evaluation

Accuracy

We select three representative DNN models as benchmarks in this paper, including Lenet (Lecun et al. 1998), Alexnet (Krizhevsky, Sutskever, and Hinton 2012) and VGG-16 (Simonyan and Zisserman 2014), see Table 1. The three DNN models are designed for two different datasets: Lenet is for MNIST, Alexnet and VGG-16 are for ImageNet. Particularly, MNIST consists of 60,000 individual images (28×28 grayscale) of handwritten digits (0-9) for training and 10000 digits for testing. ImageNet ILSVRC-2012 includes large images in 1000 classes and is split into three sets: training set (1.3M images), validation set (50K images), and testing set (100K images). The classification performance is evaluated using two measures: the top-1 and top-5 error. The former reflects the error rate of the classification and the latter is often used as the criterion for final evaluation.

Comparing our SNN with original DNN, the accuracy loss is trivial in Lenet and Alexnet (0.12% to 0.46%/0.5%). This illustrates that using the new activation function (11) does not have a dramatic impact on network accuracy. And it's unexpected that an accuracy increasement of 1.69%/0.83% is obtained in VGG-16. We attribute this phenomenon to the newly proposed activation function and this proves that it might to some extent enhance the robustness of the original network and the effectiveness of preventing over-fitting.

Dataset	Network depth	DNN err. (%)	Previous SNN err. (%)	Our SNN err. (%)
Lenet [MNIST]	11	0.84	0.56 (Rueckauer et al. 2017)	**0.92**
Alexnet [ImageNet]	20	42.84/19.67	48.2/23.8 (Hunsberger 2018)	**43.3/20.17**
VGG-16 [ImageNet]	38	30.82/10.72	50.39/18.37 (Rueckauer et al. 2017)	**29.13/9.89**

Table 1: Accuracy results.

Dataset	DNN_{mult}	DNN_{add}	SNN_{mult}	SNN_{add}	$\frac{DNN_{mult}}{SNN_{mult}}$	$\frac{DNN_{add}}{SNN_{add}}$	$\frac{DNN_{total}}{SNN_{total}}$
Lenet [MNIST]	2239k	2235k	1920k	3194k	1.17×	0.7×	0.875×
Alexnet [ImageNet]	357M	357M	39M	377M	9.15×	0.95×	1.72×
VGG-16 [ImageNet]	14765M	14757M	1496M	15505M	9.87×	0.95×	1.74×

Table 2: Evaluation on the number of operations.

The experiment results also show that our methods outperform the previous SNN architecture especially in large networks. The error rate of our SNN in Lenet is 0.36% larger and this is trivial compared with the accuracy gap in complex tasks. When processing complex tasks, our methods achieves a significant improvement in accuracy of SNN, as Alexnet (4.9%/3.63%) and VGG-16 (21.26%/8.48%). This proves that limiting the firing number of each neuron to only one spike will not reduce the accuracy of the SNN. Applying the proposed ticking neuron mechanism, the performance of temporal-coding based SNN is much better than those rate-coding based SNNs. All the conversion results on three benchmarks show that temporal-coding based SNN now is able to achieve competitive accuracy comparing with either DNN or rate-coding based SNN.

Computation Cost

To evaluate the number of operations in the networks during the entire forward propagation, We separately evaluate the amount of operations for addition and multiplication in layers, including *CONV, POOL* and *IP*. In addition to the existing multiplication and addition operations in vector dot products or potential accumulations, we also put those comparison operations into account. Each comparison operation in the *POOL* layer is treated as an addition operation and the leak of LIF neurons at each time step is considered as a multiplication operation.

The comparison of operations between DNN and our SNN ($L = 5$, which is the one with highest accuracy) is shown in Table 2. Obviously, SNN reduces the multiplication operations on all three benchmarks but also increases the number of add operations. The amount of multiplications in DNN is $1.17\times/9.15\times/9.87\times$ that of SNN in Lenet/Alexnet/VGG-16. This is because the number of multiplication in SNN is only related to the total presentation time and it will not increase with the number of pre-synaptic neurons, so the reduction of multiplications will be more significant in larger networks. But for addition operations, SNN brings additional operations due to the *ticking neuron* with high spike frequency, resulting in that the addition operations in DNN is $0.7\times/0.95\times/0.95\times$ that in SNN in the three benchmarks. Combining the multiplications and addition operations, we obtain the results on total operations. For smaller network, such as Lenet, the total number of operations in SNN is $1.14\times$ that of DNN. The reason is that the number of pre-synaptic neurons is so small that the increasement of operations brought by the *tikcing neurons* becomes significant. For larger networks, such as Alexnet and VGG-16, the computation benefits are obvious. SNN reduces the number of total operations by 41.9%/42.6%, which proves that our SNN can obtain significant computation reduction in larger networks.

Leaky Constant L

Our conversion is a much simpler and more convenient way comparing with other rate-coding based SNNs. The previous rate-coding based SNN conversion methods need to determine various important parameters, including the maximum spike frequency, spiking thresholds of each layer, etc. To determine these parameters causes huge labour and unstable performance of the converted SNN. However, in our approach, all the parameters are carefully selected according to the theoretical analysis before. Among them, the leak constant L is our primary concern.

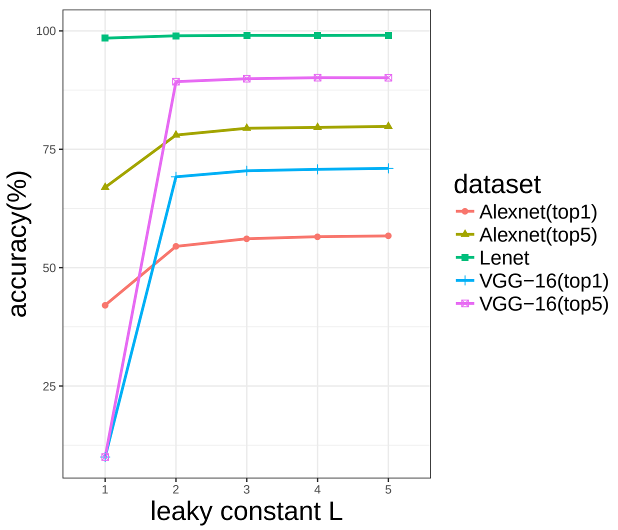

Figure 4: Effects of leaky constant L on accuracy.

As the output of equation (10), the exact firing time will increase along with the leaky constant becoming larger.

Therefore, a large L will increase the presentation time T_{il} of each layer and mapping input stimulus into more concise spike timing, which brings better performance. Choosing a small L may narrow the presentation time, resulting in divergent loss in the DNN with the adjusted architecture. The presentation time will also affect the computation overhead of the converted SNN as the *ticking neuron* fires constantly.

We evaluate the value of leaky constant L effect on the accuracy, shown in the figure 4. We find that, for the Lenet, when the L changes from 1 to 5, the accuracy has little change, just from 98.47% to 99.08%. This is because in simple tasks as MNIST dataset, the network could stand much more degraded input values. However, for the Alexnet and VGG-16, there is a significant change with the increasement of L. When L changes from 1 to 5, the accuracy of Alexnet changes from 42.08%/66.96% to 56.7%79.83% and VGG-16 changes from 0%/0%(training failure) to 70.87%/90.11%. The adjusted DNN can not converge when L is small enough as it shows in VGG-16, resulting in a collapse of the entire network.

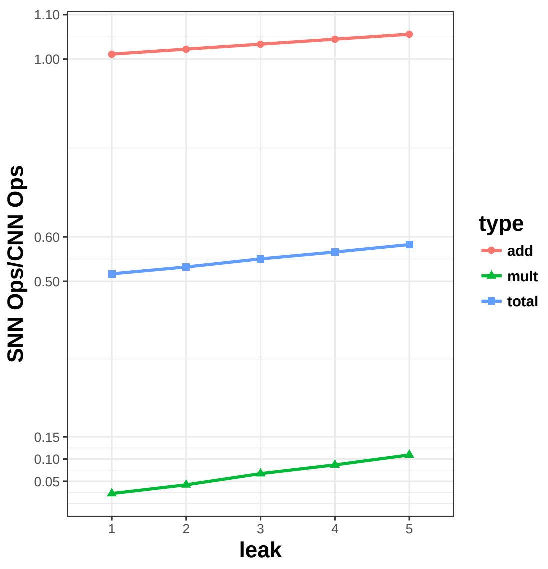

Figure 5: Effects of leaky constant L on $\frac{DNN_{Ops}}{SNN_{Ops}}$ in Alexnet.

Correspondingly, the effects of leaky constant L on the computation cost are evaluated, as shown in Figure 5. The number of addition, multiplication and total operations all increase with a larger L. When L changes from 1 to 5, the computation reductions change from 48.4% to 41.7%. This is consistent with our previous analysis that with the L increases, the increasements of presentation time will only increase the number of operations brought by *ticking neuron*.

Discussion

Considering deploying temporal-coding based SNN in SNN hardwares, even if the proposed temporal-coding based SNN can already be fully deployed on a SNN chip or platform, it may be not necessary to completely map the entire network to complete a task. In fact, only the compute-intensive layers in SNN need to be deployed, those layers require less vector productions such as max-pool could be calculated in other components of a chip or processing units. What's more, some biological mechanisms could be abandoned to accelerate computation.

Note that *ticking neuron* plays a role to excite post-synaptic neurons biologically, resulting in an interpretable SNN model. However, according to the evaluation in the last section, the deployment of *ticking neuron* will in some extent enlarge the computation cost. Thus to accelerate computing and reducing memory cost, it could be removed when deploying SNN layers in a hardware. This also allows the trained DNN weight model to be directly converted to a SNN weight model without the need of negated processing and determinations for other parameters such as ω_{tick} and ω_k.

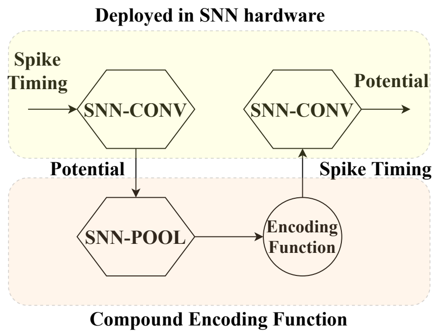

Figure 6: Deploy compute-intensive layers in SNN hardware.

Operations	DNN	SNN
Addition	$(M-1)\cdot N$	$(M-1)\cdot N$
Multiplication	$M\cdot N$	$N\cdot T_o$

Table 3: Computation cost comparison between DNN and SNN deployed in hardware theoretically.

For example, as it shows in figure 6, the *CONV* layers will be computed in a SNN hardware and those post-synaptic neurons will accumulate potential and inhibited to fire. Those layers between two synapse-based layers including activations will be considered as a compound coding function encoding the accumulated potential into spike timing for the next synapse-based layer and the selected encoding function is equation (10). After calculation, only the spike timing of the pre-synaptic neurons in the next synapse-based layers need to be stored in on-chip memory. This deployment method has a larger potential to reduce computing than the proposed SNN on SNN hardwares though a large part of interesting biological mechanisms are discarded.

Conclusion

In this paper, we propose a new conversion method along with a novel coding principle called *Reverse Coding* and a novel *Ticking Neuron* mechanism. Our methods can convert DNNs to temporal-coding SNNs with little accuracy loss and the converted temporal-coding SNNs are consistent with biological models. Based on our experiments, the presented SNNs could significantly reduce computation cost, and they are potentially alternative cost-saving models when deployed in SNN hardwares.

Acknowledgments

This work is partially supported by the National Key Research and Development Program of China (under Grant 2017YFA0700902, 2017YFB1003101), the NSF of China (under Grants 61472396, 61432016, 61473275, 61522211, 61532016, 61521092, 61502446, 61672491, 61602441, 61602446, 61732002, and 61702478), the 973 Program of China (under Grant 2015CB358800), National Science and Technology Major Project (2018ZX01031102), the Transformation and Transfer of Scientific and Technological Achievements of Chinese Academy of Sciences (KFJ-HGZX-013) and Strategic Priority Research Program of Chinese Academy of Sciences (XDBS01050200).

References

Abdel-Hamid, O.; Mohamed, A. R.; Jiang, H.; and Penn, G. 2012. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 4277–4280.

Bohte, S. M.; Kok, J. N.; and Han, A. L. P. 2000. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48(1):17–37.

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

Diehl, P.; Neil, D.; Binas, J.; Cook, M.; Liu, S.-C.; and Pfeiffer, M. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. 770–778.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; and Sainath, T. N. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Hodgkin, A. L., and Huxley, A. F. 1990. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Bulletin of Mathematical Biology* 52(1-2):25–71.

Hunsberger, E. 2018. Spiking deep neural networks: Engineered and biological approaches to object recognition. *UWSpace*.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *Eprint Arxiv*.

Koch, C., and Segev, I. 1998. *Methods in Neuronal Modeling: From Ions to Networks*. MIT Press.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. 1097–1105.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images.

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

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

Maass, W. 1997. Network of spiking neurons: the third generation of neural network models. *Transactions of the Society for Computer Simulation International* 14(4):1659–1671.

Perezcarrasco, J. A.; Zhao, B.; Serrano, C.; Acha, B.; Serranogotarredona, T.; Chen, S.; and Linaresbarranco, B. 2013. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate-coding and coincidence processing. application to feed forward convnets. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 35(11):2706–2719.

Rueckauer, B., and Liu, S. C. 2018. Conversion of analog to spiking neural networks using sparse temporal coding. In *IEEE International Symposium on Circuits and Systems*, 1–5.

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

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; and Bernstein, M. 2014. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Sainath, T. N.; Mohamed, A. R.; Kingsbury, B.; and Ramabhadran, B. 2013. Deep convolutional neural networks for lvcsr. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 8614–8618.

Severyn, A., and Moschitti, A. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *The International ACM SIGIR Conference*, 373–382.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *Computer Science*.

Thorpe, S.; Delorme, A.; and Rullen, R. V. 2001. Spike-based strategies for rapid processing. *Neural Networks* 14(6):715–725.

Van, R. R., and Thorpe, S. J. 2001. Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural Computation* 13(6):1255–1283.

Zhang, T.; Zeng, Y.; Zhao, D.; and Shi, M. 2018. A plasticity-centric approach to train the non-differential spiking neural networks.