SpikeConverter: An Efficient Conversion Framework Zipping the Gap between Artificial Neural Networks and Spiking Neural Networks

Fangxin Liu\textsuperscript{1,2}, Wenbo Zhao\textsuperscript{1,2}, Yongbiao Chen\textsuperscript{1}, Zongwu Wang\textsuperscript{1}, Li Jiang\textsuperscript{1,2,3†}

1. Shanghai Jiao Tong University, 2. Shanghai Qi Zhi Institute
3. MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
\{liufangxin, zhaowenbo, lijiang.cs\}@sjtu.edu.cn

Abstract

Spiking Neural Networks (SNNs) have recently attracted enormous research interest since their event-driven and brain-inspired structure enables low-power computation. In image recognition tasks, the best results achieved by SNN so far utilize ANN-SNN conversion methods that replace activation functions in artificial neural networks (ANNs) with integrate-and-fire neurons. Compared to source ANNs, converted SNNs usually suffer from accuracy loss and require a considerable number of time steps to achieve competitive accuracy. We find that the performance degradation of converted SNN stems from the fact that the information capacity of spike trains in transferred networks is smaller than that of activation values in source ANN, resulting in less information being passed during SNN inference.

To better correlate ANN and SNN for better performance, we propose a conversion framework to mitigate the gap between the activation value of source ANN and the generated spike train of target SNN. The conversion framework originates from exploring an identical relation in the conversion and exploits temporal separation scheme and novel neuron model for the relation to hold. We demonstrate almost lossless ANN-SNN conversion using SpikeConverter for a wide variety of networks on challenging datasets including CIFAR-10, CIFAR-100, and ImageNet. Our results also show that SpikeConverter achieves the abovementioned accuracy and the widest applicability, even on large datasets such as ImageNet (e.g., only 66.23% on CIFAR-10 (Srinivasan and Roy 2019)). On the other hand, supervised methods represented by error backpropagation with surrogate functions can achieve better performance than the unsupervised ones, but they still can not provide compatible results with ANNs in large-scale datasets (Liu et al. 2021a).

Another way to obtain parameters is by directly adapting the parameters of ANNs into SNNs, known as ANN-SNN conversion methods (Han and Roy 2020; Han, Srinivasan, and Roy 2020; Wozniak et al. 2020; Deng and Gu 2021; Li, Zeng, and Zhao 2021). These methods are devoted to finding an equivalent representation between activation values in ANN and a certain property of the spike trains. Its network structure is usually the same as the source ANN, and network parameters are transformed from source ANNs by simple operations such as scaling. The conversion methods can utilize state-of-the-art methods for training ANN to construct ANN-converted SNNs and achieve competitive accuracy and the widest applicability, even on large datasets such as ImageNet (Sengupta et al. 2019; Kim et al. 2021).

However, existing ANN-converted SNN methods are still far from applicable due to the following reasons. 1) Converted SNN still suffers from accuracy drop compared to the source ANN. ANN-converted SNNs typically exploit the firing rate of the spike train to serve as the equivalent of the activation value. However, the firing rate has far worse resolution than the activation values in ANN, leading to accuracy drop. 2) The converted SNNs need an enormous number of time steps for better information rep-
representation, which directly deteriorate the energy efficiency of SNNs. Although SNN is supposed to perform the event-driven asynchronous calculation that spikes happen at any time in the time window, the practical hardware operates on a clock-driven synchronous pattern that segment the time window into time steps and process the spikes in batches, resulting in the inference latency and energy consumption directly proportional to the number of time steps. In fact, converted ResNet on ImageNet requires up to 4096 time steps (Han, Srinivasan, and Roy 2020) while the energy consumption of AlexNet-converted SNN with 500 time steps is nearly $5 \times 10 \times$ higher than that of the source ANN (Singh et al. 2020). This leads to a significant performance decrease in terms of energy consumption and latency of the ANN-converted SNN (Rathi et al. 2021).

In such case, we make a deeper rethinking about ANN-SNN conversion methods via systematical modeling and experimental analysis. By designing a conversion framework called SpikeConverter, which includes the ANN-SNN identical relation exploration, novel neuron design, and workflow optimization, we zip the gap between the performance of ANN and SNN with the minimum number of time steps.

For clarity, we summarize our contributions as follows:

- We propose an identical relation between ANN activation values and the SNN spike trains, which concludes the validity of previous works and provides the cornerstone to precisely convert ANN into SNN.
- We put forward a temporal separation scheme to enable the identical relation and propose inverse-leaky integrate-and-fire (iLIF) neuron and layer-wise pipelining to implement temporal separation.
- We demonstrate the performance of SpikeConverter on deep network architectures on CIFAR10, CIFAR100, and ImageNet datasets. Our method provides better accuracy using 16 time steps, which is $32 \times 512 \times$ fewer than state-of-the-art ANN-converted SNNs across all network architectures and datasets we tested.

**Background and Related Work**

**ANN-converted SNN.** Converting ANNs directly into SNNs has been proven to be a promising approach to construct deep SNNs that can obtain sufficiently high accuracy on various tasks (Wu et al. 2019; Sengupta et al. 2019; Zhang et al. 2019; Kim et al. 2021; Han and Roy 2020; Han, Srinivasan, and Roy 2020). It combines the mature learning methods of ANN with the lower-power advantage of event-driven SNN, avoiding drastic accuracy drop caused by directly training SNN. As illustrated in Fig. 1, the basic idea of the ANN-converted SNN is to approximate the continuous activations in the ANN with the ReLU function (denoted as (2)) by the mean frequency of fired spikes under rate encoding (1) from the SNN. On the one hand, the training of the ANN-converted SNN relies on the back-propagation algorithm performed in the ANN, and thus it avoids the difficulties faced by the directly trained SNN. This ANN-converted SNN receives and processes the spike events in the inference phase (3). On the other hand, the ANN-converted SNN maintains a minimal gap with ANN development and can be achieved on deep network structures and large-scale datasets. To facilitate this conversion process, most of the previous works impose certain constraints on the source ANN model, such as limiting the bias to zero, eliminating batch normalization methods among the network layers, and adopting the average pooling to replace the max pooling, etc. This usually suffers from the accuracy loss compared to the original. Several techniques (Han and Roy 2020; Roy, Jaiswal, and Panda 2019; Kim et al. 2021) are used to mitigate the accuracy loss incurred in this conversion (e.g., introducing additional constraints on the fire frequency of neurons or synaptic, scaling synaptic, adding noise, etc.), but this also complicates the conversion process.

**Limitations of previous works.** Overall, ANN-converted SNNs (Roy, Jaiswal, and Panda 2019; Lobo et al. 2020) can convert and apply the latest research outcome from the ANN domain to the SNN domain relatively quickly, but this approach also has its inherent limitations. Through this direct method, ANN-converted SNNs typically need thousands of time steps to represent the information encoded in the spike train for completing a single inference, in addition to the drop in accuracy caused by imposing constraints on the source ANN (Rathi et al. 2020; Lee et al. 2020a). This is quite a large breakdown from the other SNNs, leading to significant latency and energy consumption opposite to the original purpose (Singh et al. 2020; Deng, Tang, and Roy 2021; Lee et al. 2020b; Zhang et al. 2020).

**Why we need a conversion methodology.** The fundamental difference between ANN and SNN is the notion of time. In the ANN, the inputs and outputs of neurons in each layer are real-valued (e.g., the inputs of the first layer are pixel values), and inference is performed through a single feed-forward pass of the network. The inputs and outputs of the neurons in SNN represent the spatio-temporal information using sparse spike events over time steps. Thus, the inference in SNNs is multiple feed-forward passes occurring at different time steps, where each pass requires the computation based on sparse spikes. This leads to latency and en-
energy consumption of the whole SNN directly proportional to the number of time steps. To obtain an accuracy close to the ANNs with minimal time steps of SNNS is the key to reaching a desirable tradeoff between accuracy and computation efficiency (i.e., latency and energy consumption).

**Approach**

In this section, we present a novel conversion framework called SpikeConverter to closely link the ANN and SNN by respecting the accuracy of ANN-converted SNN and enabling higher implementation performance. The proposed method finds the entire process of a desirable conversion between ANN and SNN, including the identical relation in conversion, the neuron function, and the forward process. In this way, the ANN-converted SNN can use as few time steps as possible to carry the information transferred among neurons and transmit the spatio-temporal information to deeper layers in the network without impacting accuracy.

**Soft-Reset Neuron**

In SNN, neurons are basic processing units and the information transmitted between neurons is carried by spike trains (Rashvand, Ahmadzadeh, and Shayegh 2021; He et al. 2020). Specifically, at each time step, the neuron collects all the input spikes \( X_i \) into the accumulated input \( I \) (Eq. 1) and integrates it into the membrane potential \( V \) (Eq. 2). After that, if \( V \) exceeds a pre-defined threshold \( V_{th} \), the neuron emits an output spike (Eq. 3) and resets the membrane potential (Eq. 4). One of the most widely adopted model is the leaky integrate-and-fire neuron. In Eq. (2), we denote

\[
V(t) = \dot{V}(t) - (1 - 1/\tau) V(t) + I(t)
\]

as the damping coefficient that decides the proportion of membrane potential that counts towards the next time step. Then, at the last time step \( T \), the total input voltage increment can be written as

\[
U_{in} = \sum_{i=1}^{T} \left( \sum_{t=1}^{T} w_i \cdot X_i[t] \right) \cdot k^{T-t}
\]

At the same time, the output voltage decrement can be represented as

\[
U_{out} = V_{th} \cdot \sum_{i=1}^{T} Y[t] \cdot k^{T-t}
\]

Thus, the membrane potential at the time step \( T \) can be calculated as \( V(T) = U_{in} - U_{out} \). In ideal circumstances, the membrane potential vanishes in the last time step, leading to the equation \( U_{in} = U_{out} \). Combined with Eq. (6) and Eq. (7), we have the ideal conversion identical relation

\[
\sum_{i} w_i \cdot \left( \sum_{t=1}^{T} X_i[t] \cdot k^{T-t} \right) = V_{th} \cdot \sum_{t=1}^{T} Y[t] \cdot k^{T-t}
\]

Compared with the ANN multiply-and-accumulation operation \( \sum_i w_i x_i = y \), we can see that

\[
\sum_{t=1}^{T} X_i[t] \cdot k^{T-t}
\]
Algorithm 1: Two-phase forward propagation

Input: Input spike trains $X_i$ and weight $w_i$
Output: Output spike train $Y$
Params: $V_{th}$ - the threshold voltage

Phase I: Membrane Potential Accumulation
for time step $t = 1$ to $T_1$
do
{Stage 1: Integrate input spikes into current}
$I(t) \leftarrow \text{Integrate}(w_i, X_i(t))$ \hspace{1em} $\triangleright$ Eq. (1)
// Integrate input spikes into current.
{Stage 2: Accumulate potential from input current}
$V_m(t) \leftarrow \text{Accumulate}(V_m(t-1), I(t))$ \hspace{1em} $\triangleright$ Eq. (2)
// Accumulate input current into potential.
end

Phase II: Output Spike Train Generation
for time step $t = T_1 + 1$ to $T_1 + T_2$
do
{Stage 3: Fire the output spike}
$Y(t) \leftarrow \text{Threshold}(V_m(t) - V_{th})$ \hspace{1em} $\triangleright$ Eq. (3)
// Determine whether fire the output spike.
{Stage 4: Reset the potential}
$V_m(t+1) \leftarrow V_m(t) - Y(t) \cdot V_{th}$ \hspace{1em} $\triangleright$ Eq. (4')
// Soft reset the potential if it fires a spike.
end

serves as an excellent counterpart of the activation value in 
ANN for both input and output with an multiplying factor of $V_{th}$. Such equivalence ensures the homogeneous representa-
tion of both input and output, which means that the infor-
mation can be transmitted totally in the form of spike trains 
without transforming to other modalities.

Previous works (Han, Srinivasan, and Roy 2020; Han and 
Roy 2020; Kim et al. et al. 2021) that uses frequency coding can 
be generalized in such identical relation. For example, the 
SNN uses non-leaky integrate-and-fire neuron, whose time 
constant $\tau = \infty$ and $k = 1$. Then, $\sum_t X_i[t] \cdot k^{T-t} = 
\sum_t X_i[t]$ represents the total firing time of the neuron, which 
is proportional to the firing frequency.

Temporal Separation

In the previous section, Eq. (8) holds under the assumption 
that the remaining membrane potential is 0 at the last time 
step. However, spiking neural networks lack the ability to re-
spond to negative membrane potentials, making the assump-
tion fails frequently. For example, for a non-leaky integrate-
and-fire neuron whose threshold voltage is 1, if the input 
voltage is $\{1, 1, -1, -1\}$, the output spike train $Y$ will be 
$\{1, 1, 0, 0\}$, leaving the membrane potential to be -2 eventu-
ally. Then, the total input voltage $U_{in} = 0$ and the total 
output voltage $U_{out} = 2$. The main reason is the synchro-
nized processing mechanism of spiking neural network that 
fires the output spikes in the the process of input spike train. 
The input voltage can be negative due to negative weights, 
but the output spike will not respond to it until the membrane 
potential is accumulates to positive values again.

Therefore, we propose the 
temporal separation scheme 
that separates the neural calculation into two phases, as is 
shown in Algorithm 1. The accumulating phase collects in-
put spikes without firing spikes and the generating phase 
fires the output spike according to the accumulated mem-
brane potential. Such scheme still follows Eq. (8) by scaling 
the threshold voltage $V_{th}' = kT_1 \cdot V_{th}$. Suppose the length of 
two phases are $T_1$ and $T_2$ respectively, the left-hand side of 
eq Eq. (8) remains the same by substituting $T$ with $T_1$ and 
the right-hand side of becomes

$$
(kT_1 \cdot V_{th}) \cdot \sum_{t=T_1}^{T_1+T_2} Y[t] \cdot k^{T_1-t} = V_{th} \cdot \sum_{t=T_1+1}^{T_1+T_2} Y[t] \cdot k^{T_1-t}
$$

There are several benefits that come along with this tem-
poral separating scheme. Firstly, in the last time step of the 
accumulating phase, the membrane potential will reach ex-
actly $V_{th}$, giving us a better chance to match $U_{in}$ and $U_{out}$. 
Specifically, if $V_{th} < 0$ at the last time step of accumulating 
phase, there will no output spike, naturally implemented the 
function of ReLU activation function. In addition, the sep-
oration decouples the output spike train from the input one. 
The length of the output spike train can be different from 
that of the input but decided according to the precision 
requirement of each layer in the network.

To implement temporal separation, we propose the 
inverse-leaky integrate-and-fire neuron to realize the calcu-
lation and the pipeline mechanism to minimize the delay.

Inverse-Leaky Integrate-and-Fire Neuron

In Eq. (5), since the $\tau$ represents the decaying time coef-
ficient, it is always a positive value larger than 1, leading 
to $k \in (0, 1)$. However, such configuration can not imple-
ment the temporal separation. In the generating phase, since 
there is no input spikes, the membrane potential will only be 
decreased by leaking or firing, making it always non-
creasing. Therefore, the output spike train will only con-
tain consecutive spikes. If the neuron fails to fire a spike in 
a certain spike, it will never spike any more since the 
membrane potential is already below the threshold and non-
creasing.

In such context, we propose our inverse-leaky integrate-
and-fire (iLIF) neuron whose damping coefficient $k$ is larger 
than 1. Unlike the LIF neuron that reduces the membrane 
potential in each time step, as shown in Fig. 3, iLIF aug-
ments it by the factor $k > 1$ so that it can output infor-
mative spike train continuously. The most important benefit
brought by utilizing iLIF neurons is that the conversion precision increases with the extension of output spike train. As shown in Eq. (10), even if the remaining membrane potential reaches its least upper bound \( V_{th}' \) at \( t = T_1 + T_2 \), its representing value is \( V_{th}' \cdot k^{T_1 - t} = V_{th}' \cdot k^{T_2} \), which shrinks with increasing \( T_2 \) when \( k > 1 \).

Meanwhile, there are some restrictions when iLIF neurons are utilized. Specifically, in order to avoid the situation that the membrane potential grows exponentially, the soft reset should always be able to reduce the membrane potential, i.e.

\[
k \cdot V - V_{th} \leq V, \quad 0 \leq V < V_{th}
\]

This gives

\[
k \leq \inf_{0 \leq V < V_{th}} \frac{V + V_{th}}{V} = 2
\]

**Pipelining of SNN**

The converted SNN adopting temporal separation inevitably needs more time steps to perform a single inference since the generating phase (Phase I) must be stalled until the accumulating phase (Phase II) is finished. Inspired by the fact that different processing units are allocated to work on different layers to minimize memory accessing costs, we propose the inter-layer direct delivery and inter-sample pipelining to minimize the time delay of a single sample and maximize the productivity for multiple samples.

Fig. 4 illustrates the inference pipeline of 3 layers in the SNN. Here x-axis is the inference timeline, the y-axis is the ordered layers and the charts show the working condition of processing units for each layer in different time. For simplicity, we set the lengths of two phases for each layer to be the same value \( T \). In time \( T \) to \( 2T \), we can see that from the middle chart that layer \( i \) is executing Phase II while the output is directly send to layer \( i + 1 \) for accumulation. The inter-layer direct delivery saves inference time for a single sample and also exempts the need to save output spike trains into registers and then read from them. In addition, sample 2 begins the accumulating phase for layer \( i \) from \( 2T \) to fully utilize the resource for layer \( i \). The average inference time for fully pipelined samples is \( 2T \), which compensates the longer time for single sample.

**Parameter Optimization**

In Eq. (6) and Eq. (10), we can see that difference between \( U_{in} \) and \( U_{out} \) is related to \( T, V_{th} \) and \( k \). While \( T \) is usually determined by accuracy and delay requirements, we look for the optimal threshold voltage and \( k \) by minimizing the conversion error

\[
E(V_{th}, k; T) = \mathbb{E}_{U_{in}} |U_{in} - U_{out}|
\]

which is an expected error under the distribution of \( U_{in} \). Since the layers share the same threshold, we represent the threshold voltage in proportion to the maximum \( U_{in} \) in the layers, which is denoted as \( U_{in, max} \). As shown in Fig. 5(b), we calculate the expected conversion error \( E \) for a given set of \( T = 6 \) and \( k = 1.6 \), where we can see that the error reaches its minimum when \( V_{th}/U_{in, max} = 0.396 \). After obtaining the minimum error for multiple sets of \( T \) and \( k \), Fig. 5(a) shows that \( k = 2 \) always generates the least error for every selection of \( T \). Therefore, we draw the conclusion that \( k = 2 \) gives us the optimal conversion.

**Results**

In this work, all experiments are performed on a 4-way NVIDIA Tesla V100 under the framework of Pytorch (Paszke et al. 2019). The performance of our ANN-SNN conversion methodology is examined using standard visual object recognition benchmarks, namely the CIFAR-10, CIFAR-100 (Krizhevsky, Nair, and Hinton 2014) and ImageNet datasets (Deng et al. 2009). We use VGG-16 (Simonyan and Zisserman 2015) for all three datasets, ResNet-20 configuration outlined in (He et al. 2016) for the CIFAR-10 and CIFAR-100 dataset and ResNet-34 for the ImageNet dataset. It is worth noting that we use the compact MobileNet-v2 (Sandler et al. 2018) on the ImageNet dataset as we think it will be helpful to demonstrate the effectiveness.
Inference Accuracy

The recent state-of-the-art ANN-SNN conversion works are provided for comparison as shown in Tables 1 to 3. To the best of our knowledge, our proposed SpikeConverter not only achieves the best SNN inference accuracy across all network structures and datasets we evaluated but also achieves the lowest conversion loss that allows information to be transmitted with the minimum number of time steps. Specifically, we achieve lossless and even better inference accuracy than ANN with the same network structure on the CIFAR-100 datasets. Inference accuracy is the average of the maximum inference accuracy independent runs with different seeds.

Inference Performance

Based on these, we estimate the inference accuracy and spike sparsity varies as the number of time steps with the best performing SNNSs to date as shown in Fig. 6.

In each figure, the x-axis is the SNN inference latency. The y-axis on the left indicates the SNN Top-1 inference accuracy, and the y-axis on the right indicates the average spike fired rate. We find that for the CIFAR-10 datasets in Fig. 6(a), when the inference time steps changes from 64 to 2048, the accuracy of TSC-SNN and RMP-SNN has
Table 3: Accuracy loss and time steps due to ANN-SNN conversion of the state-of-the-art SNNs on ImageNet dataset

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Spiking Neuron Model</th>
<th>ANN (Top-1 Acc)</th>
<th>Time Steps</th>
<th>SNN (Top-1 Acc)</th>
<th>Accuracy Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImageNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-34 (Sengupta et al. 2019)</td>
<td>IF (hard-reset)</td>
<td>70.69%</td>
<td>4096</td>
<td>65.47%</td>
<td>5.22%</td>
</tr>
<tr>
<td>ResNet-34 (Han, Srinivasan, and Roy 2020)</td>
<td>RMP (soft-reset)</td>
<td>70.64%</td>
<td>4096</td>
<td>69.89%</td>
<td>0.75%</td>
</tr>
<tr>
<td>ResNet-34 (Han and Roy 2020)</td>
<td>TSC (soft-reset)</td>
<td>70.64%</td>
<td>4096</td>
<td>69.93%</td>
<td>0.71%</td>
</tr>
<tr>
<td>ResNet-34 [This work]</td>
<td>SpikeConverter (soft-reset)</td>
<td>70.64%</td>
<td>16</td>
<td>70.57%</td>
<td>0.07%</td>
</tr>
<tr>
<td>VGG-16 (Rueckauer et al. 2017)</td>
<td>Converted-SNN (hard-reset)</td>
<td>63.89%</td>
<td>400</td>
<td>49.61%</td>
<td>14.28%</td>
</tr>
<tr>
<td>VGG-16 (Sengupta et al. 2019)</td>
<td>IF (hard-reset)</td>
<td>70.52%</td>
<td>2560</td>
<td>69.96%</td>
<td>0.56%</td>
</tr>
<tr>
<td>VGG-16 (Han, Srinivasan, and Roy 2020)</td>
<td>RMP (soft-reset)</td>
<td>73.49%</td>
<td>2560</td>
<td>73.09%</td>
<td>0.4%</td>
</tr>
<tr>
<td>VGG-16 (Han and Roy 2020)</td>
<td>TSC (soft-reset)</td>
<td>73.49%</td>
<td>2560</td>
<td>73.46%</td>
<td>0.03%</td>
</tr>
<tr>
<td>VGG-16 (Deng and Gu 2021)</td>
<td>ReLU+threshold (soft-reset)</td>
<td>73.47%</td>
<td>128</td>
<td>71.06%</td>
<td>2.41%</td>
</tr>
<tr>
<td>VGG-16 [This work]</td>
<td>SpikeConverter (soft-reset)</td>
<td>73.49%</td>
<td>16</td>
<td>73.47%</td>
<td>0.02%</td>
</tr>
<tr>
<td>MobileNet-v2 [This work]</td>
<td>SpikeConverter (soft-reset)</td>
<td>71.88%</td>
<td>16</td>
<td>71.71%</td>
<td>0.17%</td>
</tr>
</tbody>
</table>

Figure 6: Inference accuracy and spiking activity between converted SNN with SpikeConverter and the three state-of-the-art converted-SNNs using VGG-16 architecture on CIFAR-10 (a), CIFAR-100 (b) and ImageNet (c).

The VGG-16 with SpikeConverter inference on the CIFAR-100 dataset is shown in Fig. 6(b), which reaches the accuracy of 71.22% using only 16 time steps, whereas the SNN with IF neurons, RMP-SNN, TSC-SNN reaches 70.77%, 70.93%, and 70.97%, respectively, at the end of 2048 time steps. Here, both SpikeConverter and other SNN conversion methods were converted from our trained VGG-16 with top-1 inference accuracy of 71.22%. The SpikeConverter has reduced best inference accuracy (blue, orange, and black curve) using only 16 time steps, which is 128× faster than the compared baseline SNNs (i.e., the SNN with IF neuron, RMP-SNN, and TSC-SNN) that use about 2048 time steps. Note, the SpikeConverter with 8 time steps, even with the 4 time steps, can achieve better accuracy (71.06% accuracy) and faster inference (512×) over the 70.97% accuracy of TSC-SNN, 70.93% accuracy of RMP-SNN, and 70.77% accuracy of SNN (IF) using 2048 time steps. Meanwhile, the SpikeConverter attains a higher spike fired rate (from 11.07% to 14.23%) than others throughout the inference time steps, which means that the spike train is conveyed in SpikeConverter carries more spatio-temporal information than others.

The VGG-16 with SpikeConverter inference on the ImageNet dataset is shown in Fig. 6(c). SpikeConverter (green curve) achieved a better accuracy of 73.49% converted with the trained ANN, whereas the SNN with IF neurons (black curve) achieved 69.96%, RMP-SNN (orange curve) achieved 72.88%, and TSC-SNN (blue curve) achieved 73.46% using 2560 time steps. SpikeConverter (green curve) reaches an accuracy of 73.33% with only 8 time steps obtain 320× faster than the SNN with IF neurons (black curve) with the best accuracy and 128× times faster than the RMP-SNN (orange curve) with 72.05% inference accuracy that uses about 1024 time steps. SpikeConverter reaches the same accuracy of 73.33% using only 8 time steps, which is little change (nearly 2%), just from 92.79% to 93.63% and 90.35% to 93.63%. In contrast, our SpikeConverter achieves stable accuracy, only ≤ 0.6% accuracy change for VGG-16 network structure on CIFAR-10 when the inference time steps change from 4 to 16. Converted VGGNet by our SpikeConverter with 16 time steps exceeds the state-of-art SNN conversion methods with 2048 time steps.
Table 4: The comparison of computation cost between ANN and SNN deployed in hardware theoretically.

<table>
<thead>
<tr>
<th>Operation</th>
<th>ANN</th>
<th>SNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>((M - 1) \times N)</td>
<td>((M - 1) \times N \times T)</td>
</tr>
<tr>
<td>Multiplication</td>
<td>(M \times N)</td>
<td>0</td>
</tr>
</tbody>
</table>

Inference Computation Cost

In most ANN workloads, the key computational kernel is general matrix-matrix multiplications, which frequently appear during the forward pass. For a more intuitive comparison, we use the vector-matrix multiplication (VMM) to evaluate the computation cost, since vectors are a special case of the matrix. The comparison of computation between ANN and converted SNN with SpikeConverter is shown in Table 4. The dimensions vector-matrix multiplications is \((1 \times M) \times (M \times N)\). In the ANN, \((M - 1) \times N\) additions and \(M \times N\) multiplications are performed to compute a VMM. In the converted SNN with SpikeConverter, however, it is unnecessary to perform multiplication anymore, but only \((M - 1) \times N \times T\) additions, where \(T\) indicates the number of time steps. Since the activation is converted into the spike train in the SNN, the length of the spike train (i.e., the number of time steps) indicates that it requires \(T\) times inferences to be performed to finish computing the spike train.

Based on these, to further measure the efficiency of the SNN in terms of the theoretical computation cost, we use the spike activities (the number of addition operations required in the event-driven SNN implementation) as follows:

\[
\text{Spike Activities} = \sum_{i=1}^{L} \#OP_i \times SR_i \tag{14}
\]

where \(SR\) stands for spike fired rate (as shown in Fig. 6), \#OP represents the operations in the SNN, and \(L\) represents the total number of layers in the network. Note that, the lower the value of spike activities, the higher the energy efficiency of the SNN. As shown in Fig. 7, the number of addition operations performed in SNNs inference are also provided. We found computations do not greatly increase for the SpikeConverter with significantly less delay and better accuracy, and our method exceeds the SNN with IF neurons (black bar) and RMP-SNN (blue bar). We can find that the addition operation in SpikeConverter with the large time steps (e.g., \(\geq 16\)) is higher than TSC-SNN (orange bar). This is because TSC-SNN is encoded by temporal coding, which is significantly more sparse than other coding methods, such as the Time-To-First-Spike temporal coding representing the spike train with a single spike. However, with the fewer time steps (e.g., \(\leq 12\)), our SpikeConverter achieves close to the addition operations required in TSC-SNN. Considering the existing SNN simulation methodologies or implementations are time-driven execution mechanisms (Lee et al. 2021; Khodamoradi, Denolf, and Kastner 2021; Singh et al. 2020), SpikeConverter is a more suitable alternative with significantly better inference accuracy and performance over the three baseline SNNs.

Conclusion

In this paper, we propose SpikeConverter, an ANN to SNN conversion technique. It consists of a novel coding scheme, spiking neuron model, and inference process, which alleviates the ANN-SNN conversion information loss to significantly improve the latency and scalability of SpikeConverters to deep architectures with negligible accuracy loss. The experimental results show that the proposed SpikeConverter achieves better results than the three state-of-art ANN-SNN conversion techniques in terms of inference performance and inference accuracy.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (grant no. 61834006 and U19B2035) and the National Key Research and Development Program of China (2018YFB1403400). We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University.

References


Deng, L.; Tang, H.; and Roy, K. 2021. Understanding and Bridging the Gap Between Neuromorphic Computing...


