Replication of "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing"

Jinhao Dong
Peking University
Beijing, China
dongjinhao@stu.pku.edu.cn

Hao He
Peking University
Beijing, China
heh@pku.edu.cn

Abstract

We implement two ANN-to-SNN conversion approaches, model-based normalization and data-based normalization, as detailed in Diehl et al. [2015]. The two approaches are implemented using PyTorch and SpikingJelly without any artifacts from the original authors. We test our implementation on four standard MLP and CNN models trained from the MNIST dataset and Fashion-MNIST dataset. Our experimental results show that both normalization approaches remove the need of manual parameter tuning. What's more, compared with a baseline conversion approach that directly copies model parameters, model-based normalization enables higher performance with slower response time, while data-based normalization allows both faster response time and higher performance.

1 Introduction

Deep neural networks (DNNs) achieve great performance in many fields, such as computer vision and natural language processing. However, DNNs bears a problem that the computation of DNNs is intensive and brings high cost. This is because when feeding an input to a DNN, the neurons of one layer will transmit information to every neuron in the next layer. Spiking neural networks (SNNs) attract increasing researchers and achieve promising results. SNNs are biologically interpretable, which has the stronger learn ability. Because only when membrane voltage achieves the threshold, will the information be transmitted, the computation cost of SNNs is lower than DNNs. When training a SNN, we can train a ANN (analog neural network) first, and then map the weights to SNN. However, because there is no time dimension in ANN, the accuracy will suffer a loss after converting. Therefore, weight normalization is needed to reduce the loss. In Diehl et al. [2015], authors propose two normalization ways, model-based normalization and data-based normalization, respectively. We replicate the two normalization ways on four standard MLP and CNN models trained from the MNIST dataset and Fashion-MNIST dataset. Our result shows that model-based normalization achieves higher accuracy but has slower response time, while data-based normalization not only achieves higher accuracy but also has faster response time.

2 Background

2.1 Spiking neural networks

Among the current neural networks, deep neural networks (DNNs) are the most popular and achieve promising performance in many fields, such as computer vision and natural language processing. However, the computation cost of DNNs is high. Spiking neural networks (SNNs) are newly-proposed and rising neural network models, which attract more and more researchers. SNNs are more similar

to the ways human brains work than DNNs and can be interpreted from biology. Therefore, SNNs have the following advantages. First, the computation of SNNs is more efficient. In DNNs, every neuron will transmit information to the neurons in the next layer, which has high computation cost. In SNNs, the information transmission will not always happen. Only when the membrane potential reaches the threshold, will the information be transmitted. Second, because in SNNs the input is fed into the models continuously and the the output is also generated continuously, the input and output can be seen to happen pseudo-simultaneously. Third, because introducing the time dimension, SNNs is more efficient for time-varying inputs.

One model SNNs usually adopts is integrate-and-fire (IF) model. The accumulation of the membrane voltage V_{mem} is

$$\frac{dv_{mem}(t)}{dt} = \sum_i \sum_{s \in S_I} w_i \delta(t-s), \tag{1}$$

where w_i is weight of the i-th presynaptic neuron and S_i is the spike-times. The neuron receives the signals from the presynaptic neurons and the membrane voltage is increasing. When the membrane voltage reaches the spiking threshold V_{thr}, the neuron fires and generate a spike, which is sent to the neurons in the next layer. The membrane voltage is reset to V_{res}.

2.2 Relationships between ANNs using ReLUs and SNNs

There exits three relationships between ANNs using RELUS and SNNs. First, the computation process of ReLU is

$$x_i = \max(0, \sum_j w_{ij} x_j), \tag{2}$$

which can be seen a IF neuron without refractory period. There is no accumulation, if the signal from the neurons of the previous layer reaches the threshold 0, the neuron will fire. Second, in classification tasks, the maximum element of the output layer is the classification result, so scaling all the elements of the output layer will not affect the classification result. Third, because there is no bias, the ratio between the weights of the neurons and the ratio between the weight of the neuron and the threshold are the only parameters.

3 The Approach

3.1 Spiking Network Conversion

In order to get a SNN, we can train a ANN (analog neural network) first and then copy the weight of ANN to SNN. According to the relationships between ANNs using ReLUs and SNNs mentioned in Section 2.2, we can convert ANNs to SNNs accoring to the following way:

- Use ReLUs for all the activation functions in the ANN
- Set the bias of all the layers to zero
- Train ANN with backpropagation
- Map the weights of ANN to SNN
- Use weight normalization to the accuracy loss during the conversion

When converting a ANN to SNN, because ANN lacks time dimension, SNN will suffer a accuracy loss. The loss may come from two factors. First, the membrane potential cannot reach the threshold, which may result from the weights are too low. Second, the membrane potential is so high that the neuron generates more than one spike in one timestep. This is because the weights are higher than the threshold or there are many input signals in one timestep. Therefore, after mapping the weights of ANN to SNN, we should conduct weight normalization.

3.2 Weight normalization

In Diehl et al. [2015], authors propose two ways to normalize the weight, model-based normalization and data-based normalization. Model-based normalization needs only the weights of the network and

Algorithm 1: Model-Based Normalization

```
for layer in layers:
    max_pos_input = 0
    # Find maximum input for this layer
    for neuron in layer.neurons:
        input_sum = 0
        for input_wt in neuron.input_wts:
            input_sum += max(0, input_wt)
        max_pos_input = max(max_pos_input, input_sum)
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / max_pos_input
```

Figure 1: Model-based normalization.

Algorithm 2: Data-Based Normalization

```
previous_factor = 1
for layer in layers:
    max_wt = 0
    max_act = 0
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            max_wt = max(max_wt, input_wt)
        max_act = max(max_act, neuron.output_act)
    scale_factor = max(max_wt, max_act)
    applied_factor = scale_factor / previous_factor
    # Rescale all weights
    for neuron in layer.neurons:
        for input_wt in neuron.input_wts:
            input_wt = input_wt / applied_factor
    previous_factor = scale_factor
```

Figure 2: Data-based normalization.

data-based normalization needs not only the weights of the network but also the activations of each layer when feeding data. The two algorithms is in Fig. 1 and Fig. 2 respectively.

In model-based normalization, for each layer, compute the summation of the weights above zero of each neuron in this layer. And then find the maximum of the weight summations among all the neurons in this layer, all the weights are divided by this maximum. In data-based normalization, for each layer, find the maximum of all the activations and find the maximum of all the weights, and the bigger one is the current scale factor.

4 Experimental Setup

As required by this assignment, we choose to independently implement the two aforementioned ANN-to-SNN conversion approaches without using or referencing the original MATLAB implementation provided by the authors.[1] To ease implementation, we use PyTorch [Paszke et al., 2019] for dataset preparation, specification, and training of the ANN models, and we combine SpikingJelly [Fang et al., 2020] to implement and simulate their corresponding SNN models. All experiments are conducted on an Ubuntu 20.04 machine with RTX 3090.

We implement and train two ANN architectures, MLP-784-1200-1200-10o and CNN-28x28-12c5-2s-64c5-2s-10o, identical to those mentioned in Diehl et al. [2015]. The first architecture is a multi-layer preceptron (MLP) with two hidden layers, ReLU activation, and dropout layers. The second architecture is a convolutional neural network (CNN) with two convolutional layers, average

[1] `https://github.com/dannyneil/spiking_relu_conversion`

Table 1: Performance of the ANN Models

Dataset	Model	Train Accuracy	Test Accuracy
MNIST	MLP	0.9981	0.9791
	CNN	0.9979	0.9893
Fashion-MNIST	MLP	0.9344	0.8810
	CNN	0.9269	0.8950

Table 2: Performance of the Converted SNN Models. We set simulation time-step to 1ms, reset potential $v_{reset} = 0.0$, and fix all input rates to 1000 Hz instead of altering input rates as done in Diehl et al. [2015]. We report the converged final test accuracy using a simulation length of 10000ms.

Dataset	Model	Conversion Method	Threshold	Test Accuracy
MNIST	MLP	Baseline (Direct Copy)	1.0	0.9864
		Baseline (Direct Copy)	4.0	0.9868
		Model-Based Normalization	1.0	0.9865
		Data-Based Normalization	1.0	0.9866
	CNN	Baseline (Direct Copy)	1.0	0.9887
		Baseline (Direct Copy)	20.0	0.9911
		Model-Based Normalization	1.0	0.9910
		Data-Based Normalization	1.0	0.9910
Fashion-MNIST	MLP	Baseline (Direct Copy)	1.0	0.8975
		Baseline (Direct Copy)	4.0	0.9000
		Model-Based Normalization	1.0	0.8969
		Data-Based Normalization	1.0	0.8985
	CNN	Baseline (Direct Copy)	1.0	0.8933
		Baseline (Direct Copy)	20.0	0.9018
		Model-Based Normalization	1.0	0.9015
		Data-Based Normalization	1.0	0.9020

pooling, ReLU activation, and dropout layers. For easy conversion to SNN, we set all bias to zero in the two models during training.

The original paper [Diehl et al., 2015] only test their approaches on the MNIST dataset [LeCun et al., 1998], which may be too simple for today's DNN-based approaches and thus difficult to highlight the advantages of sophisticated normalization techniques. Therefore, we also include a more difficult image classification dataset, FashionMNIST [Xiao et al., 2017], in our experiment.

The source code of our implementation, execution logs, and the figures used in this report can be assessed in this GitHub link.[2]

5 Experimental Results

Table 1 reports the performance of our trained ANN models. In the MNIST dataset, both the MLP and CNN model can achieve near perfect train accuracy, but the MLP model is prone to over-fitting and perform poorly on the test set (acc. = 0.9791) compared with the CNN model (acc. = 0.9893). Similar results can be observed in the Fashion-MNIST dataset (MLP test acc. = 0.8810, CNN test acc. = 0.8950).

Table 2 reports the performance of different SNNs converted from a variety of approaches. We compare the results of model-based normalization and data-based normalization with a simple baseline conversion approach that directly copies all model parameters into the corresponding SNN. For the MNIST dataset, we obtain similar results as in the original paper [Diehl et al., 2015]. Additionally, we can also observe from Table 2 that the baseline conversion approach require manual tuning of spiking threshold to avoid performance losses, while model converted using the

[2] `https://github.com/hehao98/SNN-Balancing`

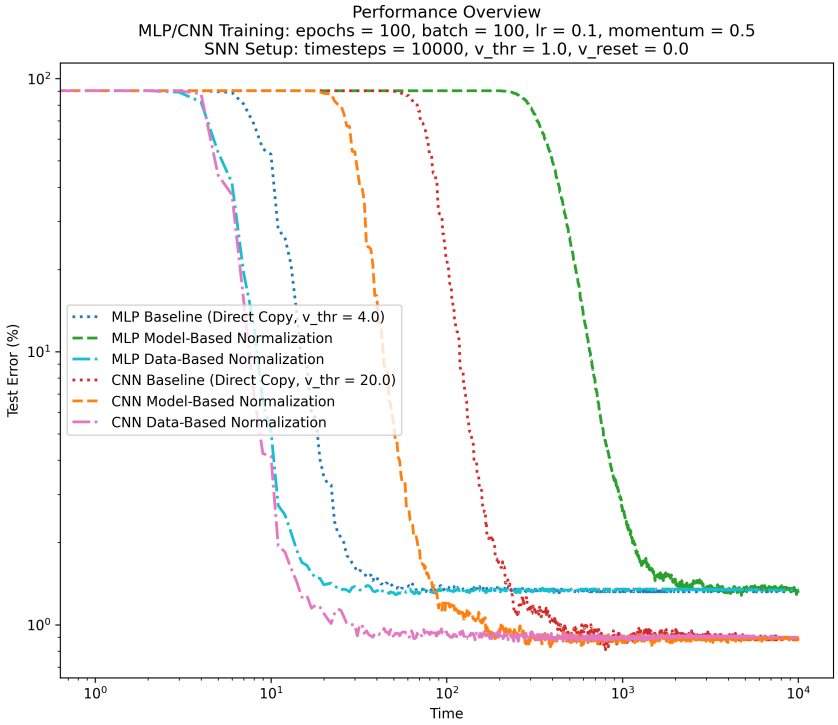

Figure 3: SNN Classification Error Over Time in the MNIST Dataset

normalization approaches can achieve high performance using a default threshold of 1.0. The above observation holds for experiments conducted on both dataset, and a data-based normalization of CNN trained on Fashion-MNIST can achieve the highest test accuracy of 0.9020 among all models.

Figure 3 shows the classification error of different converted SNNs on the MNIST dataset over a simulation time period of 10000ms. We choose such a long time period because SNNs converted using model-based normalization typically require high convergence time as the model parameters are often significantly scaled down. We obtain similar results in Figure 3 compared with Figure 3 in Diehl et al. [2015]. We can also clearly observe that models converted using data-based normalization can achieve high accuracy using the shortest simulation time (< 30ms, cyan and pink lines). These SNNs can have very low latency in specialized hardware and can be promising in real-time applications.

Similarly, Figure 4 shows the classification error of different converted SNNs on the Fashion-MNIST dataset. In the case of Fashion-MNIST, model-based normalization still requires very long time to reach peak performance while data-based normalization converges very fast. Although data-based normalization seem to be less effective for the SNN converted from MLP (cyan line), it is especially advantageous for the SNN converted from CNN: the SNN can achieve the highest test accuracy with the lowest latency (< 100ms, the pink line). Our additional experiments on Fashion-MNIST confirm the effectiveness of the data-based normalization technique proposed by Diehl et al. [2015].

6 Conclusion

In this replication report, we have described our implementation of two ANN-to-SNN conversion approaches described in Diehl et al. [2015] and the experimental results we obtained. We use an additional image classification dataset, FashionMNIST [Xiao et al., 2017], to verify the generalizability of the two approaches. Our experimental results largely align with findings in the original paper [Diehl et al., 2015], and confirm the effectiveness of data-based normalization on ANN-to-SNN conversion.

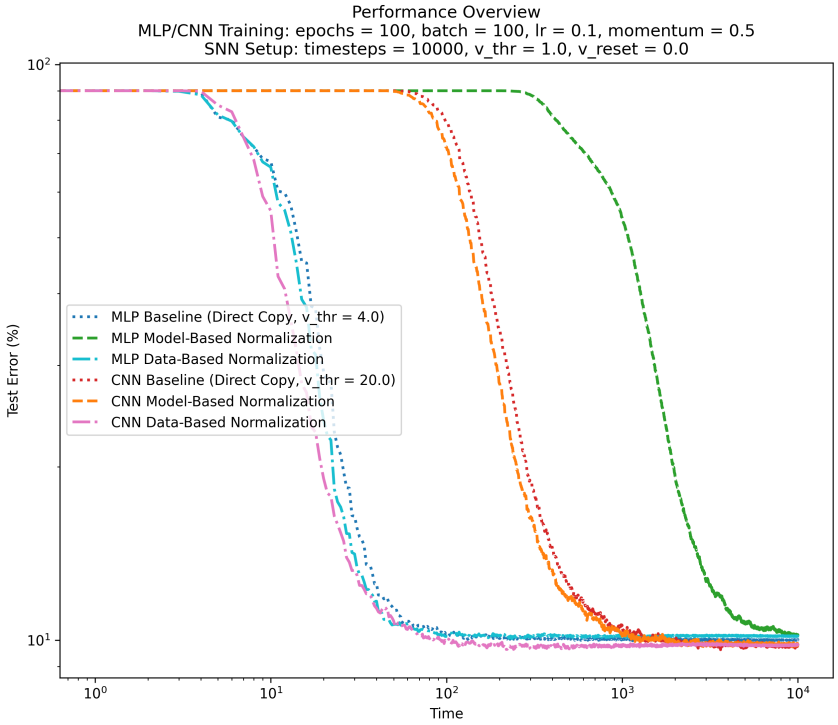

Figure 4: SNN Classification Error Over Time in the Fashion-MNIST Dataset

Acknowledgments

Both authors contribute equally in the implementation and experiments. In this report, Jinhao Dong wrote Section 1, Section 2, and Section 3; Hao He wrote Abstract, Section 4, Section 5, and Section 6.

References

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–8. IEEE, 2015. doi: 10.1109/IJCNN.2015.7280696.

Wei Fang, Yanqi Chen, Jianhao Ding, Ding Chen, Zhaofei Yu, Huihui Zhou, Yonghong Tian, and other contributors. SpikingJelly. `https://github.com/fangwei123456/spikingjelly`, 2020. Accessed: 2021-06-21.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL `http://arxiv.org/abs/1708.07747`.