Efficient Techniques for Training the Memristor-based Spiking Neural Networks Targeting Better Speed, Energy and Lifetime

Yu Ma
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
mayu@shanghaitech.edu.cn

Pingqiang Zhou
School of Information Science and Technology
ShanghaiTech University
Shanghai, China
zhoupq@shanghaitech.edu.cn

ABSTRACT

Speed and energy consumption are two important metrics in designing spiking neural networks (SNNs). The inference process of current SNNs is terminated after a preset number of time steps for all images, which leads to a waste of time and spikes. We can terminate the inference process after proper number of time steps for each image. Besides, normalization method also influences the time and spikes of SNNs. In this work, we first use reinforcement learning algorithm to develop an efficient termination strategy which can help find the right number of time steps for each image. Then we propose a model tuning technique for memristor-based crossbar circuit to optimize the weight and bias of a given SNN. Experimental results show that the proposed techniques can reduce about 58.7% crossbar energy consumption and over 62.5% time consumption and double the drift lifetime of memristor-based SNN.

CCS CONCEPTS

• **Hardware → Emerging technologies**; • **Computer systems organization → Reliability**; • **Computing methodologies → Neural networks**.

KEYWORDS

Memristor, Spiking neural network, Drift, Reinforcement learning, Model tuning

ACM Reference Format:
Yu Ma and Pingqiang Zhou. 2021. Efficient Techniques for Training the Memristor-based Spiking Neural Networks Targeting Better Speed, Energy and Lifetime. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431555

1 INTRODUCTION

The past decades have witnessed the evolution of the artificial neural networks (ANNs), which were invented to model how human brain processes data and how it learns to recognize objects [1]. Although the current ANN is able to achieve high accuracy in many applications, neurons in ANN communicate with continuous valued activations, which is the most important difference from human brain [2]. In contrast, the spiking neural networks (SNNs) adopt spikes as the information propagation method, which is more bio-mimetic. In the rest of this paper, we refer ANNs to non-spike neural networks to distinguish them from SNNs.

Speed and energy consumption are two important metrics in SNN inference process. Inputs need to be multiplied with the weights which is referred as multiply-and-accumulate (MAC) operation. Crossbar based computing is proposed to accelerate this process. In architecture level, TrueNorth [3] is proposed to accelerate SNN inference with CMOS crossbar. As an alternative way, memristor-based crossbar is proposed [4]. In order to train the weight and bias of SNN, two kinds of direct training methods – bio-inspired spike-timing-dependent plasticity (STDP) algorithm [5] and error propagation [6] – can be used. However, they are not effective with large deep networks [7]. Alternatively, many researchers propose to convert the well-trained ANN to SNN [8].

In the conversion process, weight and bias are normalized to avoid approximation errors due to too low or too high firing rate [8]. Model-based and data-based normalization [8] are firstly proposed. The former normalizes the model with theoretical maximum output value of the model while the latter uses training set to optimize the normalization process. However, normalized weight and bias of these two methods are too small which leads to low firing rate – each layer needs more time to fire a spike to the next layer. Thus, the conversed SNN has large time and energy consumption. Robust normalization [9, 10] is proposed to solve this problem. However, this method is based on empirical parameters which cannot always get the best weight and bias for minimum time and energy consumption of inference. In this work, we explore a better normalization method in the conversion process of memristor-based SNN.

In ANNs, each layer is recalled (computed) once in each image classification process. However, in SNNs, each layer is generally recalled dozens of times. Each recall operation leads to energy and time consumption. Besides, although memristors can be programmed to desired resistances, the resistance will be changed after recall operations which is known as the drift problem [11]. The accumulated change of resistance will influence the accuracy of neural networks. As a result, the more the recall times are, the higher latency, energy consumption and drift impact will be. *Therefore, if we can reduce the recall times in algorithm level, not only energy and time consumption, but also drift impact can be reduced.* In this paper, we analyze the relationship between the accuracy and the number of time steps and observe that each image needs different numbers of time steps to be classified correctly. In order to terminate the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPDAC '21, January 18–21, 2021, Tokyo, Japan
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431555

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.

classification process at different time steps for different images, we propose a novel termination strategy technique for SNN.

In our work, we propose efficient techniques for training the memristor-based SNN targeting better speed, energy and drift lifetime. The flow of the proposed ANN-SNN conversion framework is shown in Figure 1 which consists of four steps. The first two steps are conventional methods, which output an initial SNN model. Then the initial SNN model and the given training set are used to find the best termination strategy for the SNN. Finally, the trained termination strategy and the initial SNN model, together with the training set, are used to tune the weight and bias of the SNN. The contributions of this work are summarized as below:

- Based on the analysis of the relationship between accuracy and the number of time steps, *we propose an efficient termination strategy which can terminate the classification process at the right number of time steps for all images.* Therefore, the speed, energy and drift lifetime of memristor-based SNN can be optimized.
- We observe the chance of optimizing the normalization method according to memristor crossbar characteristics. *This motivates us to propose a model tuning technique which can further optimize speed, energy and drift lifetime compared with the state of the art normalization methods.*
- We evaluate our proposed techniques with a three-layer SNN network on MNIST dataset [12]. Results show that the energy and time consumption can be reduced over 58.7% and 62.5% respectively and the drift lifetime can be doubled.

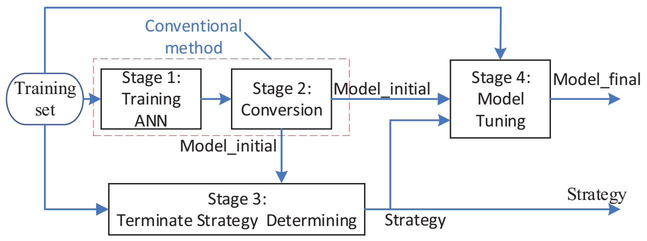

Figure 1: Framework of the Proposed Techniques.

The rest of this paper is organized as follows. In Section 2, we introduce SNN algorithm and memristor-based SNN computing. In Section 3, we present the motivation of our work. The proposed techniques are described in Section 4. At last, experimental results are described in Section 5, followed by conclusion in Section 6.

2 PRELIMINARIES

2.1 Spiking Neural Networks

Typically, there is a preset number of time steps, T_{max}, in SNN. In order to simulate neurons, integrate-and-fire (IF) model is widely used [13]. Firstly, the inputs of the SNN, such as images, are converted to spiking inputs. In each time step, neurons update their membrane potentials (MPs) as Equation (1):

$$z_i^l(t) = \sum_j w_{ij}^l \Theta_j^{l-1}(t) + b_i^l \tag{1}$$

where Θ_j^{l-1} represents the j-th spiking input of layer l, w_{ij}^l is the weight value and b_i^l is the bias of the i-th neuron in layer l. Then, the spiking output of layer l in time step t is determined by:

$$\Theta_{t,i}^l = U\left(V_i^l(t-1) + z_i^l(t) - V_{th}\right) \tag{2}$$

where $U(x)$ is a step function and $V_i^l(t-1)$ is the MP in time $t-1$. If the MP of the i-th neuron exceeds threshold V_{th}, the neuron sends a spike to the next layer. Then the MP is reset to resting potential:

$$V_i^l(t) = \begin{cases} V_{rest,i}^l & \Theta_i^l(t) = 1 \\ V_i^l(t-1) + z_i^l(t) & Otherwise. \end{cases} \tag{3}$$

The outputs of the neurons in the last layer are recorded and after the preset number of time steps, $T_{\max}$, the result corresponding to the neuron which fires most spikes will be classified as the result.

In order to get an SNN model, converting ANN to SNN is a wildly used method [8]. Firstly, an ANN is trained using ReLU (Rectified Linear Unit) for all activation units of the network. Then the trained network is mapped to an IF network and the weight and bias are normalized according to: $\mathbf{W}^l = \mathbf{W}^l/\left(\frac{\lambda^l}{\lambda^{l-1}}\right)$, $\vec{b}^l = \vec{b}^l/\lambda^l$, where λ^l is the normalization factor of layer l. After training set is propagated through the ANN, the ReLU activations are stored. [8] proposes data-based normalization, using the maximum of ReLU activation ($\lambda^l = \max[a^l]$) as the normalization factor. Furthermore, [9, 10] propose robust normalization by modifying λ to p-th percentile of the total activity distribution, $\lambda^l = percentile[a^l]$. The authors in [9, 10] empirically find that 99-th or 99.9-th percentile is the best normalization method.

2.2 Memristor Crossbar Based SNN Computing Circuit

Crossbar based neural processing unit (NPU) is considered as an alternative classification accelerator [13] which is shown in Figure 2. Each NPU includes crossbars, peripheral circuits and controllers. The crossbar is used to receive the output of the previous layer. The horizontal lines of the crossbar represents axons of previous layer and vertical lines represent the dendrites of the current layer.

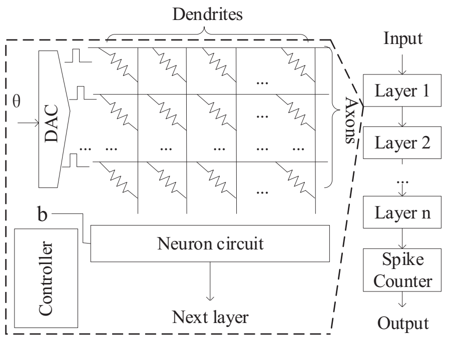

Figure 2: Crossbar Based Computing Circuit.

Weights of one layer ($\mathbf{W}^l$) are mapped to memristor [14] crossbars' conductance $\mathbf{G}$ while the inputs (Θ^{l-1}) are converted to input voltages $\vec{V}$ by digital-to-analog converter (DAC). The crossbar is able to perform vector-matrix multiplication according to Kirchhoff's law which is a MAC operation: $\vec{I} = \mathbf{G} \cdot \vec{V}$. Then the $\vec{I}$ and bias ($\vec{b}^l$) are integrated in the peripheral neuron circuit performing Equation (2) and (3). Then the result is transferred to the next layer, and spike counter is used to record the output of the last layer.

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.

Ideally, the resistance change of memristor happens if and only if the voltage applied to the device exceeds a certain threshold in programming phase. In practice, however, the resistance of memristors can also be changed slowly by low voltage pulses in recall processes [15] which is termed as drift. Therefore, the weights which are stored in memristors are changed when drift happens. This leads to the accuracy degradation of the inference.

3 MOTIVATION OF OUR WORK

3.1 Analysis of Preset Time Steps

When an SNN model is adopted for the classification process, the number of spikes of the whole SNN is directly proportional to the number of time steps. Besides, number of spikes directly influences the energy and drift degree of memristor-based crossbar circuit. Therefore, a direct way to improve SNN's performance (including latency, energy and drift), is to reduce the number of time steps. Therefore, how to set the number of time steps is an important research problem. We test an SNN converted with method[8] and the relationship between accuracy and number of time steps on MNIST data set are shown in Figure 3. *The blue line "Waste" denotes the redundant time steps after the identified image has been able to be correctly classified.*

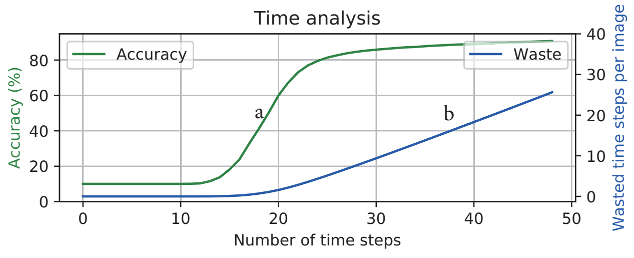

Figure 3: Time Analysis. (a) accuracy, (b) wasted time steps.

No matter how to set the number of time steps, there is a waste of time for those images who need fewer time steps. Images, who need more time steps, are not able to be classified correctly. Therefore, there doesn't exist an optimal time step for all images. Besides, the wasted time steps will lead to extra latency, energy consumption and drift problem. As a result, the best termination strategy should be image-wise which is able to get both high accuracy and low consumption. Motivated by this, we explore a novel termination strategy technique for SNNs called "WINNER K" which will be introduced in Section 4.1.

3.2 Analysis of Normalization Methods

Assume we are able to find a method which is able to terminate the classification process for all images once the image classification result is the same as the preset number of time steps, the next question is – can we tune the weight and bias of SNN model to further reduce drift degree and energy and time consumption? To analyze this question, we test how the model normalization affects the total number of spikes and time consumption of an SNN model, and the results are summarized in Figure 4. In this experiment, we assume that we terminate the classification process once we can get the result which is the same as the result with preset number of time steps (50 in this experiment).

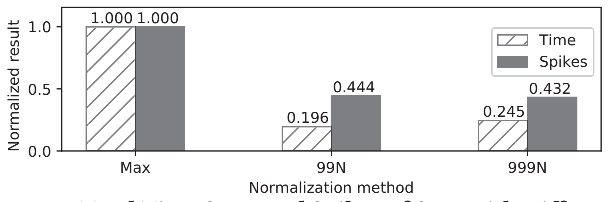

Figure 4: Total Time Steps and Spikes of SNN with Different Normalization Methods.

As shown in Figure 4, we test three normalization methods, data-based normalization (noted as Max) and two robust normalization methods (99-th and 99.9-th percentile normalization which are noted as 99N and 999N respectively). Model with the data-based normalization method has the largest time consumption and total number of spikes because it's too conservative. Each neuron in hidden layers needs the largest spikes input before reaching the threshold to produce a spike to the next layer. Both 99-th and 99.9-th normalization have a huge improvement compared with data-based normalization on both total number of spikes and time steps. Besides, in this experiment, the model with 99-th normalization has higher classification speed while 99.9-th normalization can save more number of spikes. The result shows that even the best models have their own benefit and victim. If we want to optimize the model for a certain purpose, it's necessary to tune the weight and bias according to circuit characteristics.

4 PROPOSED TECHNIQUES

4.1 Termination Strategy

In conventional SNNs, a preset number of time steps (T_{max}) is used to terminate the classification process. If we set the number of time steps to T_{max}, fewer than T_{max} time steps is enough for the SNN to correctly classify most images as discussed in Section 3. This motivates us to find a strategy to terminate the classification process image by image. With a small number of time steps (T_0), if one image can be classified, activation of different neurons in the output layer are different. Otherwise, the activations of neurons in the output layer are similar. Considering this, we propose "WINNER K" method which explores to terminate the classification process according to difference of neurons' activation in the output layer. Assume that in the output layer, one neuron produces the maximum number of spikes, SO_0, and another neuron produces the second maximum number of spikes, SO_1. We monitor $SO_0 - SO_1$ and terminate the classification process once $SO_0 - SO_1 = K$. In this way, different images can be terminated with different numbers of time steps in the classification process. The reason is that in a classification task, if an output neuron produces K more spikes than others at time step T, it may be in the lead of other neurons since then. As a result, if the classification process is terminated at this time step, the result will be the same as the result classified with T_{max} time steps. As an output neuron leads more time steps, the probability of classifying correctly is larger while the time consumption is also larger. In order to find the best K in our strategy, we apply reinforcement learning [16] method which is an acclimatized method to find a strategy based on samples. We construct a state transition map of the problem as shown in Figure 5. The states of this problem are as follows:

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.

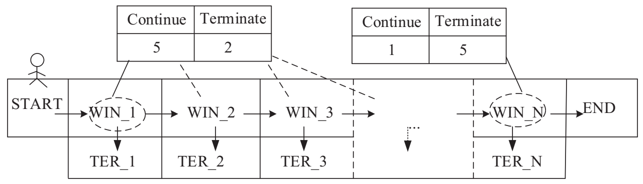

Figure 5: State Transition Map.

- *START* is the initial state of the problem, which represents the state at time step 0. At this state, the image is started to be classified. The SNN has done nothing and the reward is 0 at this state. The agent starts at this state.
- *WIN_n* is the intermediate state which means $SO_0 - SO_1 = n$. The reward of the agent at this state is the time penalty.
- *TER_n* represents that the SNN is terminated once it reaches the *WIN_n* state. At this state, the SNN inference is terminated and classification result, R_n, is returned. The agent gets the reward according to R_n and the ground truth. If R_n is the same as the ground truth, the reward will be R_c (correct reward) and otherwise, R_p (fault penalty).
- *END* is the state which the number of time steps reaches the preset value, $T_{\max}$. At this state, the classification process is forced terminated and the reward of the agent is also determined depending on the consistency of the result and the ground truth. Note that, as the agent reaches this state with some time after the previous state *WIN_n*, the reward should add the time penalty.

Algorithm 1 Algorithm of Strategy Making.

```
Input: States $\mathcal{S}$, Actions $\mathcal{A}$, Time limit $T$, Learning rate $\alpha$, Discount
    factor $\gamma$, Exploration factor $\epsilon$, $R_c$, $R_p$.
Output: Strategy, $\mathcal{X}$
 1: for all Images do
 2:     if Converged then
 3:         Break
 4:     end if
 5:     $reward \leftarrow 0$
 6:     for $t = 0$ to $T$ do
 7:         $reward \leftarrow reward - penalty$;
 8:         if Reach the new $WIN\_N$ state then
 9:             Update Q table and reset reward;
10:             Determine next state with $\epsilon$ and Q table;
11:             if $nextstate == TER$ then
12:                 Break
13:             end if
14:         end if
15:     end for
16:     Compute reward with $R_c$ and $R_p$;
17:     Update Q table and $\epsilon$;
18: end for
19: return Strategy $\mathcal{X}$ according to Q table.
```

The pseudo-code of the method is shown in Algorithm 1. For each image, we start the test process at the *start* state ($t = 0$) in line 6. For each time step, we simulate the normal SNN algorithm. Line 7 records the time penalty for every time step. Line 8 judges whether the agent reaches *WIN_k* states. If so, the agent updates the Q table in line 9 and then determines the next state as a sample in line 10. The agent chooses a direction randomly with probability ϵ and chooses the direction depending on the Q table with probability $1 - \epsilon$. If the agent chooses to go to the next state, $WIN\_k + 1$, the SNN algorithm continues. Otherwise, if the agent chooses to go to the state *TER_k*, the SNN is terminated, and the Q table is updated in line 16 and 17. If the number of time steps reaches T_{max}, the agent reaches the state *END*. Meanwhile, the SNN and the Q table are also terminated and updated respectively. The Q table is updated according to Equation (4).

$$Q(s,a) = (1-\alpha)Q(S,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right) \quad (4)$$

where $R(s, a, s')$ is the reward of this state, $Q(s, a)$ is Q value of the action a at the state s.

The agent learns strategy image by image with the training set. After the Q table converges, the learning process is stopped (line 2). Then we go through each item of the Q table and determine the strategy as *WINNER_K* once "continue" value is smaller than "terminate" value at the state *WIN_K* (line 19). For example, values in the Q table are shown in Figure 5. For *WIN_1* to $WIN\_(N-1)$, "continue" value, 5, is larger than "terminate" value, 2. The strategy maker goes on until the *WIN_N* state, where the "continue" value, 1, is smaller than the "terminate" value, 5. In this case, the parameter K of the strategy is determined as N and the strategy is determined as *WINNER_N*. For every image, the SNN classification process will be terminated once $SO_0 - SO_1 = N$. The classification for each image reaches this state with different number of time steps. Thus, we are able to terminate the classification process at the right number of time steps for each image.

4.2 Model Tuning

As analyzed in Section 3, the weight and bias directly influence the total number of spikes and time steps in the classification process. Besides, the more spikes produced in the classification process, the higher energy consumption and drift degree will be. After finding a termination strategy with an initial SNN model, we optimize the weight and bias with a tuning parameter $\mathcal{P}$ as shown in Equation (5).

$$\mathbf{W}^l = \mathbf{W}^l_{ini} \cdot \mathcal{P}_l \qquad \vec{b}^l = \vec{b}^l_{ini} \cdot \mathcal{P}_l \quad (5)$$

where $\mathbf{W}^l_{ini}$ and $\vec{b}^l_{ini}$ are weight and bias in layer l of the initial SNN model which is used to train the termination strategy, $\mathcal{P}_l$ is the tuning factor of layer l and $\mathcal{P} = \{\mathcal{P}_l\}$. In order to find the best parameter $\mathcal{P}$, we construct a loss function according to characteristics of memristor-based circuits.

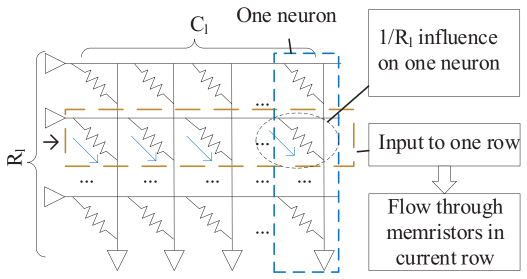
C_l
One neuron
$1/R_l$ influence on one neuron
Input to one row
Flow through memristors in current row
R_l

Figure 6: Influence of Input in One Layer.

As shown in Figure 6, R_l and C_l are numbers of rows and columns of layer l respectively. In our proposed technique, we define S_l as

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.

the number of input spikes of layer l and its value is determined by testing a batch of images in training set. In addition, if we tune parameter $\mathcal{P}$, the S_l will be changed. Therefore, S_l is a function of $\mathcal{P}$. The loss function is associated with the energy consumption, which contains two aspects. 1) Each spike input will lead to current flow through memristors which are in the same row as the input. Thus, each spike input will lead C_l memristors to consume energy. 2) Total number of spikes is also influenced by the number of rows. If the number of rows of one layer is large, S_l will be large. If we only use the first aspect to build the loss function, the layer which has larger number of rows will be optimized much more than other layers, which influences the accuracy and speed of SNN. As a result, in order to balance the optimization degree of each layer, we need to normalize the energy consumption with R_l. Thus, the loss function is constructed as Equation (6):

$$loss = \sum_{l} C_l * S_l(\mathcal{P})/R_l \tag{6}$$

Equation (6) is also suitable to optimize the drift problem. Each spike input to layer l drives current flow through C_l memristors which leads to drift problem. Therefore, we need to punish S_l with C_l. On the other hand, there are R_l rows in layer l, each spike input will change $1/R_l$ number of weights in each column. Assuming that each weight has the same influence on each neuron, each spike input leads to $1/R_l$ error of each neuron in layer l. Considering these two aspects, C_l and $1/R_l$ should be used to punish the S_l in drift problem.

Based on the aforementioned analysis, we can optimize SNN by optimizing the constructed loss function as shown in Equation (6). We tune the factor $\mathcal{P}$ to reduce the loss, the pseudo-code of our technique is shown in Algorithm 2. Besides, it's still difficult to find the best $\mathcal{P}$ because there is variation in the classification process. In order to solve this problem and make the technique more robust, we use the medium loss value to represent the loss of current $\mathcal{P}$ in line 21. The proposed technique performs tuning process (line 2-18) $N \times R$ times. After every R times of tuning, β is reduced in line 17 to tune $\mathcal{P}$ more accurately. In each tuning process, the technique tunes one factor at a time and l is to label the current tuning factor of $\mathcal{P}$. The proposed technique first chooses an initial tuning direction for the current tuning factor (line 3). Then it computes whether the loss is reduced following this direction in line 5 - 8. $\mathcal{P}_{next}$ only changes the l-th factor of $\mathcal{P}$, $\mathcal{P}_{next,l} = \mathcal{P}_l(1+\beta)$. If the loss is reduced, we continue tuning the $\mathcal{P}$ following this direction until the loss can't be reduced. Otherwise, we try to find a better $\mathcal{P}$ in the opposite direction in line 14. If we are not able to get a better $\mathcal{P}$ in both directions, we move to the next parameter, $\mathcal{P}_{l+1}$, in line 12.

Algorithm 2 Algorithm of Normalization Tuning.

```
Input: Tuning parameter $\mathcal{P}$ with size $L$, Tuning rate $\beta$, Step reduction factor $\alpha$, Reduction times $N$, Repeat times $R$.
Output: Optimized normalization parameter $\mathcal{P}$
 1: function TUNING($\mathcal{P}$)
 2:     for $i = 0$ to $N$, $j = 0$ to $R$, $l = 0$ to $L$ do
 3:         $go \leftarrow True$; $d \leftarrow 1$;
 4:         loop
 5:             $loss_{current} \leftarrow$ MEDIUM($\mathcal{P}, \beta$);
 6:             Determine $\mathcal{P}_{next}$ with $d$ and $\beta$;
 7:             $loss_{next} \leftarrow$ MEDIUM($\mathcal{P}_{next}$);
 8:             if $loss_{next} < loss_{current}$ then;
 9:                 $go \leftarrow True$; $\mathcal{P} \leftarrow \mathcal{P}_{next}$;
10:             else
11:                 if $go == False$ then
12:                     break;
13:                 end if
14:                 $d \leftarrow -1 \cdot d$; $go \leftarrow False$;
15:             end if
16:         end loop
17:         $\beta \leftarrow \alpha\beta$;
18:     end for
19:     return $\mathcal{P}$
20: end function
21: function MEDIUM($\mathcal{P}, \beta$)
22:     for all $\{\mathcal{P} \cdot (1-\beta), \mathcal{P}, \mathcal{P} \cdot (1+\beta)\}$ do
23:         record and find medium loss
24:     end for
25:     return medium loss
26: end function
```

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

Following the practice of previous work which study memristor drift problem [15], we evaluate the efficiency of our proposed techniques on MNIST dataset [12]. In our experiment, we use memristor model in [17] with four bit quantization and tune the parameter to quick drift similar to [15] in order to speed up the simulation. In our experiment, the number of 20 time steps is able to maintain over 95% accuracy. Thus, we set $T_{max} = 20$. Network and parameters that are used in this work are shown in Table 1. Besides, we assume that the time consumption is directly proportional to the number of time steps. Five cases are evaluated in our experiment as follow:

- Case 1 (99N): 99-th percentile normalization model [10] without our proposed techniques.
- Case 2 (999N): 99.9-th percentile normalization model [10] without our proposed techniques.
- Case 3 (99T): 99-th percentile normalization model with our proposed termination strategy.
- Case 4 (999T): 99.9-th percentile normalization model with our proposed termination strategy.
- Case 5 (OT): Weight and bias are tuned with our proposed tuning technique and the termination strategy is applied.

Table 1: Network and Parameters.

layer 1	layer 2	layer 3	Accuracy
784×1000	1000×30	30×10	98.11%

R_c/R_p	α	γ	R_t	α	β	N	R
±50	0.1	0.9	−1	0.2	0.5	4	2

5.2 Experimental Results

5.2.1 Static evaluation. The accuracy and time consumption per image of the converted SNN are shown in Figure 7. The experimental results show that the accuracy of 99.9-th percentile normalization is a bit lower than that of 99-th normalization because of

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.

the higher firing rate. The tuned model has the highest accuracy because of the highest firing rate. Besides, the 99N and 999N have almost the same accuracy with 99T and 999T respectively.

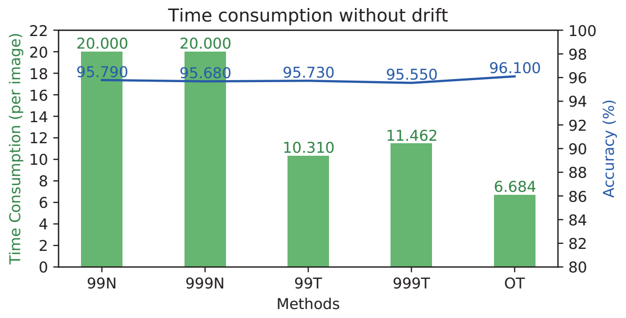

Figure 7: Time Consumption and Accuracy without Drift.

For each image, the models 99N and 999N need 20 time steps to complete the classification task. With the proposed termination technique, the average time consumption for images' classification is decreased by 48% and 42% respectively compared with models 99N and 999N. The reason why 99N model can be optimized a bit more than 999N is that the former model has higher firing rate which leads to less time consumption. Besides, the tuned model only needs fewer than 7 time steps per image which is decreased by 35.17% and 41.69% compared with models 99T and 999T respectively. This shows that the proposed model tuning technique is efficient for SNN weight normalization. Totally, our proposed techniques can reduce about 70% time steps.

5.2.2 Drift evaluation. Considering the drift phenomenon, we evaluate the lifetime of the memristor-based SNN, which is defined as the number of image classifications before the accuracy degrades below 85%. We analyze two indicators: one is drift lifetime, the other is energy and time consumption. Note that the energy consumption simulated in our experiment is the crossbar energy.

The result of the accuracy degradation is shown in Figure 8. The proposed termination technique can increase about 66% and 48% lifetime of memristor-based SNN with 99-th and 99.9-th normalization respectively. The proposed model tuning technique can further increase about 15% lifetime. Totally, the proposed techniques can almost double the lifetime compared with conventional methods.

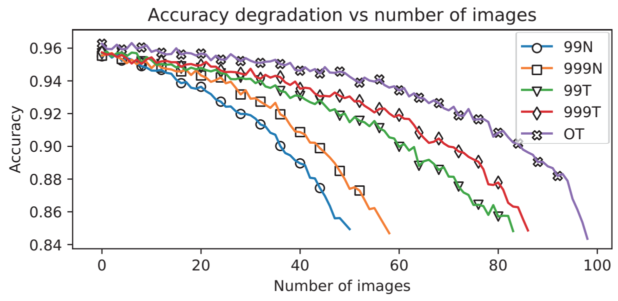

Figure 8: Drift Lifetime.

In addition, we monitor the latency and energy consumption of each image during the classification process, and the average consumption is shown in Table 2. The time steps consumption of the five models are similar to the results in static evaluation. The results of the crossbar energy consumption show that our proposed techniques can reduce this consumption significantly. Our techniques can reduce up to 58.7% crossbar energy consumption compared with previous methods. Besides, the energy consumption of peripheral circuits is related to the number of spikes and time consumption. As a result, our proposed techniques can also reduce the energy of peripheral circuits which is not evaluated in our current work.

Table 2: Lifetime, Average Time Consumption and Average Energy Considering Drift.

Method	99N	999N	99T	999T	OT
Time steps per Image	20.00	20.00	11.35	12.10	7.50
Lifetime (# Images)	50	58	83	86	98
Crossbar energy (μJ)	18.85	19.48	10.10	11.91	8.04

6 CONCLUSION AND FUTURE WORK

In this paper, we propose two efficient techniques for training the memristor-based SNNs with the objective of optimizing speed, energy and drift lifetime. After analyzing the relationship between accuracy and the number of time steps, we propose an efficient termination strategy for SNNs. Furthermore, observing that the inference time and the number of spikes are also affected by the way normalizing the SNN in the conversion process, we construct a loss function to tune the weight and bias of SNN. The experimental results show that compared with conventional methods, the proposed techniques can reduce up to 62.5% time consumption and 58.7% energy consumption and can double the drift lifetime of memristor-based SNN. In our future work, we will test deeper convolutional neural networks on larger data sets.

REFERENCES

[1] F. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65 6, pp. 386–408, 1958.
[2] A. Tavanaei *et al.*, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47 – 63, 2019.
[3] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
[4] T. Tang *et al.*, "Spiking neural network with rram: Can we use it for real-world application?" in *DATE*, 2015, pp. 860–865.
[5] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: A comprehensive overview," *Frontiers in Synaptic Neuroscience*, vol. 4, p. 2, 2012.
[6] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.
[7] S. Park, S. Kim, H. Choe, and S. Yoon, "Fast and efficient information transmission with burst spikes in deep spiking neural networks," in *DAC*, 2019, pp. 1–6.
[8] P. U. Diehl *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *IJCNN*, 2015, pp. 1–8.
[9] B. Rueckauer *et al.*, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *ArXiv*, 2016.
[10] B. Rueckauer *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017.
[11] C. Ting, J. Sung-Hyun, and L. Wei, "Short-term memory to long-term memory transition in a nanoscale memristor," *ACS Nano*, vol. 5, no. 9, pp. 7669–7676, 2011.
[12] L. Yann, C. Corinna, and C. J. Burges, "The mnist database of handwritten digits," [Online], http://yann.lecun.com/exdb/mnist.
[13] E. Vatajelu, G. Di Natale, and L. Anghel, "Special session: Reliability of hardware-implemented spiking neural networks (SNN)," in *VTS*, 2019, pp. 1–8.
[14] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
[15] B. Yan *et al.*, "A closed-loop design to enhance weight stability of memristor based neural network chips," in *ICCAD*, 2017, pp. 541–548.
[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Int. Res.*, vol. 4, no. 1, p. 237–285, May 1996.
[17] J. P. Strachan *et al.*, "State dynamics and modeling of tantalum oxide memristors," *IEEE Transactions on Electron Devices*, vol. 60, no. 7, pp. 2194–2202, 2013.

Authorized licensed use limited to: ShanghaiTech University. Downloaded on June 05,2021 at 04:51:57 UTC from IEEE Xplore. Restrictions apply.