SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Area-efficient memristor spiking neural networks and supervised learning method

Errui ZHOU[1], Liang FANG[1*], Rulin LIU[1] & Zhensen TANG[1]

[1] *Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha* 410073*, China*

Appendix A Mathematical models of synapses and neurons

Appendix A.1 Synapse model

In conventional neural networks, there are weights with positive and negative values. However, memristor conductance is always positive in hardware implementation. In this letter, we adopt the scheme where each synapse comprises two memristors, as shown in Figure 1. These two memristors are connected to the same output port of the pre-synaptic neuron but are connected to different input ports of the post-synaptic neuron. One memristor is chosen to represent the positive weight, which equals the conductance in value. The other represents the negative weight, which equals the conductance in absolute value but has a negative sign. Therefore, the weight of a synapse can be expressed as follows:

$$\omega = G^{+} - G^{-}. \tag{A1}$$

The maximum and minimum values of G^{+} and G^{-} are G_{max} and G_{min}, respectively, and G_{max} is usually hundreds times greater than G_{min}. Hence, ω has a range of $[G_{min} - G_{max}\ ,\ G_{max} - G_{min}]$. Given that the weights of SNNs usually have a large range, memristors with large conductance ranges are required. Here we assume the memristor shows a conductance range of $[10^{-7}S, 10^{-4}S]$, which is practical in experiments. Therefore, the weight range of a synapse is $[(10^{-7} - 10^{-4})S, (10^{-4} - 10^{-7})S]$.

Appendix A.2 Neuron model

We define the voltages integrated on the positive port and the negative port as $V^{+}(t)$ and $V^{-}(t)$, respectively. When the voltage difference between $V^{+}(t)$ and $V^{-}(t)$ exceeds a predefined threshold(i.e., the neuron is activated), a spike shaped step signal is generated by the neuron, which can be expressed by (A2) and (A3):

$$V(t) = V^{+}(t) - V^{-}(t), \tag{A2}$$

$$V_{out} = \begin{cases} VDD, & V(t) >= V_t, \\ 0, & otherwise, \end{cases} \tag{A3}$$

where VDD is the supplied voltage that does not change the conductance of memristors, and V_t is the predefined threshold. When the step function is represented by $\varepsilon(x)$, (A3) can be rewritten as follows:

$$V_{out} = VDD \cdot \varepsilon(V(t) - V_t). \tag{A4}$$

When the activation time of the neuron is noted as t_j, (A4) also can be expressed as follows:

$$V_{out} = VDD \cdot \varepsilon(t - t_j). \tag{A5}$$

We approximate the current flowing through a memristor as a product of voltage and conductance, which can be express as $i = V \times G$. Then charges integrated on a capacitor can be expressed as $Q = it$, and the integrated voltage can be expressed as Q/C.When there are several pre-synaptic neurons connected to a post-synaptic neuron via synapses, the integrated voltages can be represented by following equation:

$$V^{+}(t) = \frac{VDD}{C} \sum_i G_i^{+} \cdot (t - t_i) \cdot \varepsilon(t - t_i), \tag{A6}$$

* Corresponding author (email: lfang@nudt.edu.cn)

$$V^-(t) = \frac{VDD}{C}\sum_i G_i^- \cdot (t - t_i) \cdot \varepsilon(t - t_i), \tag{A7}$$

where C represents the capacitance of integrated capacitors, and $\{t_i, i = 0, 1 \cdots\}$ are activation times of pre-synaptic neurons. We then get (A8) according to (A1), (A2), (A6), and (A7). The schematic of temporal changes of the integrated voltage is shown in Figure A1. The integrated voltage changes with a certain rate when the positive current $i1$ is applied, the integrated voltage changes with a slower rate when the negative current $i2$ is applied, the integrated voltage changes with a faster rate when the positive current $i3$ is applied, and the integrated voltage exceeds the threshold voltage finally.

$$V(t) = V^+(t) - V^-(t) = \frac{VDD}{C}\sum_i (G_i^+ - G_i^-) \cdot (t - t_i) \cdot \varepsilon(t - t_i) = \frac{VDD}{C}\sum_i \omega_i \cdot (t - t_i) \cdot \varepsilon(t - t_i), \tag{A8}$$

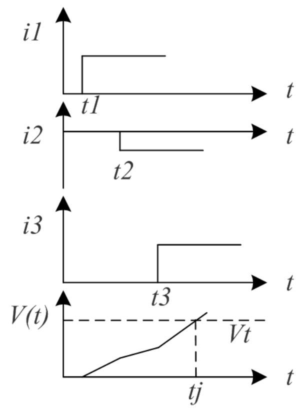

Figure A1 Temporal changes of neurons.

Spikes with specific shapes need more complex generators and controllers, e.g., spikes in [11] have two additional parameters: delay constant and decay constant. However, our neurons need no extra parameters. By simplifying spikes of the memristor SNNs to step signals, an integrator and a comparator are enough for hardware implementation of a neuron, which decreases design complexities and costs compared with previous studies. Hardware implementations of existing literatures and this work are shown in Table A1.

Table A1 Comparison of different hardware implementations

	Integrator	Comparator	Controller	Spike generator
Wu [1]	√	√	√	√
Park [2]	√	√	\	√
Yu [11]	√	√	√	√
This paper	√	√	\	\

Appendix B Supervised learning method

To train the above memristor SNNs by error backpropagation, there should be a set of inputs and desired outputs. In the above memristor SNNs, the desired outputs are defined as the desired activation times of output neurons, which can be noted as t_d. The actual outputs are noted as t_o. Therefore, we can define the error function as follows:

$$E = \frac{1}{2}\sum_{o \in O}(t_o - t_d)^2. \tag{B1}$$

The weight adjustment of the synapse between the $o_m th$ output neuron and the $h_n th$ hidden neuron is computed by following equation:

$$\Delta\omega_{o_m h_n} = -\eta \frac{\partial E}{\partial \omega_{o_m h_n}}, \tag{B2}$$

where η is the learning rate, which is positive, and $\partial E/\partial\omega_{o_m h_n}$ is the gradient. We decompose the gradient as follows:

$$\frac{\partial E}{\partial \omega_{o_m h_n}} = \frac{\partial E}{\partial t_{o_m}} \cdot \frac{\partial t_{o_m}}{\partial V_{o_m}(t_{o_m})} \cdot \frac{\partial V_{o_m}(t_{o_m})}{\partial \omega_{o_m h_n}}. \tag{B3}$$

The first and third right-hand factors are easy to get as follows:

$$\frac{\partial E}{\partial t_{o_m}} = t_{o_m} - t_{d_m}, \tag{B4}$$

$$\frac{\partial V_{o_m}(t_{o_m})}{\partial \omega_{o_m h_n}} = \frac{VDD \cdot (t_{o_m} - t_{h_n}) \cdot \varepsilon(t_{o_m} - t_{h_n})}{C}. \tag{B5}$$

The second right-hand factors can be expressed as (B6) according to [5].

$$\frac{\partial t_{o_m}}{\partial V_{o_m}(t_{o_m})} = -\frac{1}{\frac{\partial V_{o_m}(t_{o_m})}{\partial t_{o_m}}}. \tag{B6}$$

We follow the derivation process of SpikeProp, therefore, Eqs. (B1)-(B6) are almost the same as SpikeProp except for the step function. However, the step function is not differentiable, which prevents the computation of the gradient. To compute the gradient, we approximate the step function as a modified sigmoid function, which is differentiable. The modified sigmoid function is expressed by following equation:

$$f(t - t_i) = \frac{1}{1 + e^{-s(t-t_i)}}, \tag{B7}$$

where t_i is the step time and s is used to modified the function shape. When we set $s = 10^6$, curves of the modified sigmoid function and the step function are shown in Figure B1. We can find that the modified sigmoid function curve is almost the same as the step function curve in the range of milliseconds. Hence, it is reasonable to replace the step function by the modified sigmoid function. However, the modified sigmoid function is differentiable, which can be used in computation of gradient. Then (B6) is rewritten to (B8). We obtain the gradient as (B9) by computing (B4), (B5), (B8) together. Therefore, the weight adjustment is expressed as (B10).

$$\frac{\partial t_{o_m}}{\partial V_{o_m}(t_{o_m})} = -\frac{C}{VDD \cdot \sum_{h \in H} \omega_{o_m h} \frac{(1+e^{-s(t_{o_m}-t_h)}) + se^{-s(t_{o_m}-t_h)}(t_{o_m}-t_h)}{(1+e^{-s(t_{o_m}-t_h)})^2}}. \tag{B8}$$

$$\frac{\partial E}{\partial \omega_{o_m h_n}} = -\frac{(t_{o_m} - t_{d_m})(t_{o_m} - t_{h_n})}{(1 + e^{-s(t_{o_m}-t_{h_n})}) \sum_{h \in H} \omega_{o_m h} \frac{(1+e^{-s(t_{o_m}-t_h)}) + se^{-s(t_{o_m}-t_h)}(t_{o_m}-t_h)}{(1+e^{-s(t_{o_m}-t_h)})^2}}. \tag{B9}$$

$$\Delta\omega_{o_m h_n} = \frac{\eta(t_{o_m} - t_{d_m})(t_{o_m} - t_{h_n})}{(1 + e^{-s(t_{o_m}-t_{h_n})}) \sum_{h \in H} \omega_{o_m h} \frac{(1+e^{-s(t_{o_m}-t_h)}) + se^{-s(t_{o_m}-t_h)}(t_{o_m}-t_h)}{(1+e^{-s(t_{o_m}-t_h)})^2}}. \tag{B10}$$

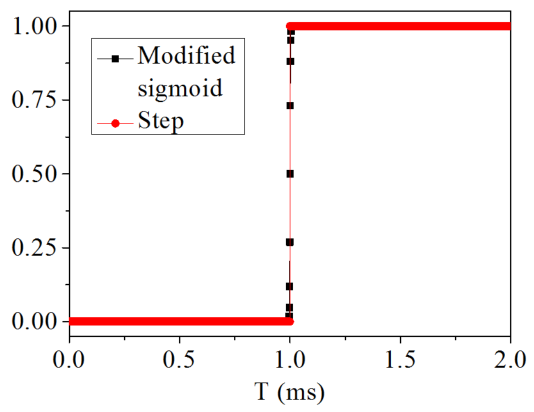

Figure B1 Curves of the step function and the modified sigmoid function.

When the output neuron spikes after the hidden neuron, $t_{o_m} - t_{h_n} > 0$. Since the denominator of (B10) is positive, then $\Delta\omega_{o_m h_n} > 0$ when $t_{o_m} - t_{d_m} > 0$ and $\Delta\omega_{o_m h_n} < 0$ when $t_{o_m} - t_{d_m} < 0$. This means that the weight increases when the actual spiking time of the output neuron is later than the desired time, and the weight decreases when the actual spiking time of the output neuron is earlier than the desired time. When the output neuron spikes before the hidden neuron, which means the hidden neuron has little influence on the output neuron, the weight is almost unchanged. From the analysis of (B10), the weight can be adjusted correctly to make the actual time equal to the desired time finally.

Assume there is only one hidden layer, the weight adjustment of the synapse between the $h_m th$ hidden neuron and the $i_n th$ input neuron is computed by following equation:

$$\Delta\omega_{h_m i_n} = -\eta \frac{\partial E}{\partial \omega_{h_m i_n}}. \tag{B11}$$

The gradient can be expressed as (B12), which can be computed similarly. Therefore, (B11) can be rewritten as (B13).

$$\frac{\partial E}{\partial \omega_{h_m i_n}} = \sum_{o \in O} \frac{\partial E}{\partial t_o} \cdot \frac{\partial t_o}{\partial V_o(t_o)} \cdot \frac{\partial V_o(t_o)}{\partial t_{h_m}} \cdot \frac{\partial t_{h_m}}{\partial V_{h_m}(t_{h_m})} \cdot \frac{\partial V_{h_m}(t_{h_m})}{\partial \omega_{h_m i_n}}. \tag{B12}$$

$$\Delta\omega_{h_m i_n} = \eta \sum_{o\in O}(t_o - t_d)\cdot \frac{\omega_{oh_m}\frac{(1+e^{-s(t_o-t_{h_m})})+se^{-s(t_o-t_{h_m})}(t_o-t_{h_m})}{(1+e^{-s(t_o-t_{h_m})})^2}}{\sum_{h\in H}\omega_{oh}\frac{(1+e^{-s(t_o-t_h)})+se^{-s(t_o-t_h)}(t_o-t_h)}{(1+e^{-s(t_o-t_h)})^2}} \cdot \frac{\frac{(t_{h_m}-t_{i_n})}{(1+e^{-s(t_{h_m}-t_{i_n})})}}{\sum_{i\in I}\omega_{h_m i}\frac{(1+e^{-s(t_{h_m}-t_i)})+se^{-s(t_{h_m}-t_i)}(t_{h_m}-t_i)}{(1+e^{-s(t_{h_m}-t_i)})^2}}. \tag{B13}$$

The weight adjustment of the synapse between the $h_m th$ hidden neuron and the $i_n th$ input neuron is more complex because the final sign is depended on $t_o - t_d$, ω_{oh_m}, and $t_{h_m} - t_{i_n}$. However, it can train the SNNs to correct direction. Though we only consider the SNNs with one hidden layer, the computation is similar when there are multiple hidden layers, which is not shown in this paper.

Appendix C Simulation results

Appendix C.1 The XOR problem

The XOR problem is a classic nonlinear problem which is usually employed as a benchmark to test SNNs [5,7]. The inputs and outputs of the XOR problem are binary. There are 4 groups of inputs: $\{1, 1\}$, $\{0, 0\}$, $\{0, 1\}$, and $\{1, 0\}$, and their corresponding outputs are 0, 0, 1, and 1 respectively. The simulation time is 15ms, and the time step is 0.01ms. The capacitance of the capacitor is 1nF, the supplied voltage is 1V, and the predefined threshold is 0.8V. The spiking times of inputs and desired outputs after temporal-coding are shown in Table C1. The input neuron spikes at 0 ms when its corresponding input is 1, and it does not spike (i.e., the spiking time is ∞) when its corresponding input is 0. There is a bias neuron that is always activated in the input layer and the hidden layer respectively. Therefore, the memristor SNN for the XOR problem has three input neurons (one bias neuron), three hidden neurons (one bias neuron) and two output neurons. Differ with [5], there are two output neurons representing two kinds of outputs respectively and there is a bias neuron in hidden layer. The numbers of neurons and synapses employed in the memristor SNN and existing literatures are shown in Table C2. The numbers of neurons employed in different works are almost the same. However, the numbers of synapses vary greatly because the numbers of sub-connections vary. Our memristor SNN only employs 12 synapses while [5] and [11] employ 320 synapses and [7] uses 80 synapses, which means our memristor SNN is more area-efficient.

Table C1 Spiking times of inputs and desired outputs.

	inputs			outputs
a1	a2	a3	b1	b2
$0ms$	$0ms$	$0ms$	$10ms$	∞
∞	∞	$0ms$	$10ms$	∞
$0ms$	∞	$0ms$	∞	$10ms$
∞	$0ms$	$0ms$	∞	$10ms$

Table C2 Neurons and synapses employed in different algorithms for XOR.

	Input neuron	Hidden neuron	Output neuron	sub-connection	Synapse
Bohte [5]	3	5	1	16	320
Ghosh-Dastidar [7]	3	5	1	4	80
Yu [11]	3	5	1	16	320
This paper	3	5	2	1	12

Learning results of the memristor SNN for the XOR problem are shown in Figure C1. The learning goal is that the memristor SNN converges to a mean squared error (MSE) of 0.25. It can be seen that the memristor SNN converge fastest when the learning rate is 0.005 and it achieves the goal at the 30th epoch. It is obvious that the memristor SNN converges more rapidly than [5, 7, 11], according to Table C3. Additionally, our learning method and [11] are proposed for hardware implementation while [5] and [7] are for software implementation.

Table C3 Learning results of different algorithms for XOR.

	Learning rate	MSE for convergence (ms^2)	No. of epochs for convergence
Bohte [5]	0.01	0.25	250
Ghosh-Dastidar [7]	0.005	0.5	78
Yu [11]	\	1	2500
This paper	0.005	0.25	30

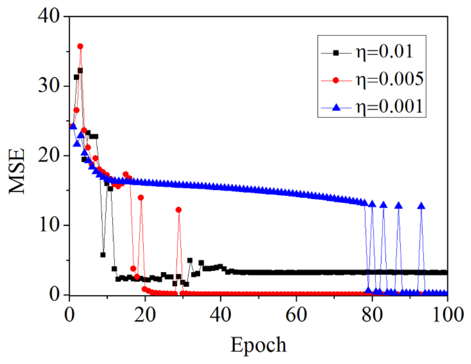

Figure C1 Convergence curves of the XOR problem.

Appendix C.2 The Fisher Iris dataset

The classification problem of the Fisher Iris dataset is more practical than the XOR problem, which classifies three classes by four features. The dataset contains 150 samples (each class has 50 samples). 75 samples (25 samples for each class) are selected randomly as training set and the remaining samples are employed as testing set. However, features are expressed by real values rather than logic values, which means features should be converted to binary. By analyzing the values of four features, we employ eight bits to express each feature (three bits for the integer part and five bits for the decimal part). Therefore, the memristor SNN for the Fisher Iris dataset has 33 input neurons (one bias neuron) and three output neurons. Then, the number of hidden neurons is selected as nine (one bias neuron). The simulation time, step time, the capacitance of the capacitor, the supplied voltage, and the predefined threshold are consistent with the configurations in the XOR problem. The input neurons spike at 0 ms when their corresponding inputs are 1, and they do not spike when their corresponding input are 0. Similarly, when the data belong to a class, the corresponding output neuron spikes at 10 ms, and others do not spike. The number of neurons employed in the memristor SNN is close to [5–7], as shown in Table C4. However, the synapses used in the memristor SNN are several times less than [5–7], which means the memristor SNN is more area-efficient. Note that less neurons and synapses will be employed if input neurons are reduced properly.

Table C4 Neurons and synapses employed in different algorithms for Fisher Iris.

	Input neuron	Hidden neuron	Output neuron	sub-connection	Synapse
Bohte [5]	50	10	3	16	8480
Xu [6]	21	8	3	5	960
Ghosh-Dastidar [7]	17	8	3	4	640
This paper	33	9	3	1	291

Convergence curves with different learning rates are shown in Figure C2. The learning goal is that the memristor SNN converges to a MSE of 0.25. It can be seen that the memristor SNN converges fastest when the learning rate is 0.005 and it achieves the goal at about the 200th epoch. Details of different learning algorithms are shown in Table C5. Our learning algorithm is faster than [6] but slower than [7]. However, [7] decomposes the three-class problem into three two-class problems, which decreases training epochs but increases classification steps. What's more, the memristor SNN can classify all classes correctly after training. Classification accuracies of different algorithms are shown in Table C6. Obviously, our learning method obtains the highest accuracy compared to existing literatures.

Table C5 Learning results of different algorithms for Fisher Iris.

	Learning rate	MSE for convergence (ms2)	No. of epochs for convergence
Xu [6]	0.0001-0.0004	0.4	234
Ghosh-Dastidar [7]	0.01	0.2	60
This paper	0.005	0.25	200

Appendix C.3 The MNIST dataset

The MNIST dataset contains 70000 grayscale images of handwritten digits. Each image contains 28×28 pixels. The training set has 60000 images, and the testing set comprises the remaining images. In training and testing, each pixel corresponds to an input neuron. For simplicity, pixels of all images are binarized to two classes: pixels with grayscale values that are not lower than 15 generate spikes at time $t = 0$, pixels with lower grayscale values do not generate spikes. Here the spike is the step signal as above. The number of hidden layers is chosen as one, and the number of hidden neurons is selected as

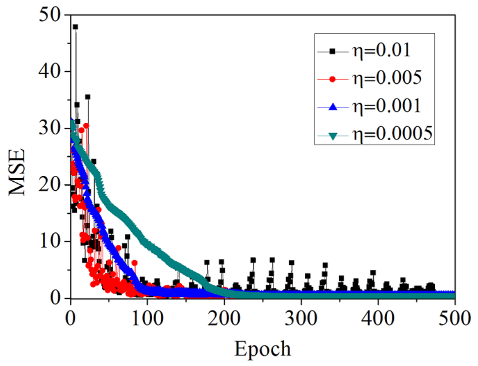

Figure C2 Convergence curves of the Fisher Iris.

Table C6 Classification accuracies of different algorithms.

	Training set(%)	Testing set(%)	Total(%)
Lin [3]	98.1	96.7	97.4
Sporea [4]	96	94	95
Bohte [5]	97.4	96.1	\
Xu [6]	99.96	94.44%	95.54
Ghosh-Dastidar [7]	\	\	95.87
This paper	100	100	100

101(one bias neuron). Similarly, there is a bias neuron in the input layer. Therefore, the structure of the memristor SNN is $785 - 101 - 10$. Note that bias neurons in the input layer and the hidden layer generate spikes at time $t = 0$. Simulation setup is the same as the Fisher Iris dataset.

Table C7 Performance results and comparison with other learning methods.

	Architecture	Encoding type	Learning type	Learning rule	Accuracy(%)
Kheradpisheh [8]	Convolutional SNN	Spike-based	Unsupervised	STDP	98.4
Diehl [9]	Two layer network	Spike-based	Unsupervised	STDP	95.0
Zhao [10]	Convolutional SNN	Spike-based	Supervised	Tempotron rule	91.3
Lee [12]	Convolutional SNN	Rate-based	Supervised	Back-propagation	99.31
Diehl [13]	Convolutional SNN	Rate-based	Supervised	Back-propagation	99.12
Hussain [14]	Dendritic neurons	Rate-based	Supervised	Morphology learning	90.3
O'Connor [15]	Spiking RBM	Rate-based	Supervised	Contrastive divergence	94.1
Zhang [16]	Voltage-driven three-layer	Spike-based	Supervised	Equilibrium and STDP	98.52
This paper	Three layer network	Spike-based	Supervised	Back-propagation	97.61

Performance results and comparison with other learning methods are shown in Table C7. Note that the learning methods compared are not SpikeProp-like. We can find that the highest accuracy is given by the convolutional, rate-based SNN in [12]. However, rate-based SNNs usually generate amount of spikes, and require many time steps to get an answer [8]. As we all know, convolutional ANNs usually get better performances than feedforward fully connected ANNs. SNNs also have similar phenomena. The highest accuracy with STDP learning method is reported in [16]. The accuracy of our paper is 97.61%, slightly lower than 98.52% in [16]. Though the performance of our memristor SNN is not the highest, the significant advantage of our memristor SNN and learning method is that they are designed for hardware implementation. What's more, our paper only uses one hidden layer with 101 hidden neurons while other literatures employ several hidden layers or thousands of neurons.

References

1 Wu X, Saxena V, Zhu K, et al. A CMOS Spiking Neuron for Brain-Inspired Neural Networks With Resistive Synapses and In Situ Learning. IEEE Trans Circuits Syst II Exp Briefs, 2015, 62: 1088–1092

2 Park J, Kwon M W, Kim H, et al. Compact Neuromorphic System With Four-Terminal Si-Based Synaptic Devices for Spiking Neural Networks. IEEE Trans Electron Devices, 2017, 64: 2438–2444

3 Lin X, Wang X, Hao Z. Supervised learning in multilayer spiking neural networks with inner products of spike trains. Neurocomputing, 2017, 237: 59–70

4 Sporea I, Gruning A. Supervised Learning in Multilayer Spiking Neural Networks. Neural Comput, 2013, 25: 473–509

5 Bohte S M, Poutre J A L, Kok J N. Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing, 2002, 48: 17–37

6 Xu Y, Zeng X, Han L, et al. A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks[J]. Neural Networks, 2013, 43: 99–113

7 Ghosh-Dastidar S, Adeli H. A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural Networks, 2009, 22: 1419–1431

8 Kheradpisheh S R, Ganjtabesh M, Thorpe S J, et al. STDP-based spiking deep convolutional neural networks for object recognition. Neural networks, 2018, 99: 56–67

9 Diehl P U, Cook M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front Comput Neurosc, 2015, 9: 1–9

10 Zhao B, Ding R, Chen S, et al. Feedforward Categorization on AER Motion Events Using Cortex-Like Features in a Spiking Neural Network. IEEE Trans Neural Netw Learn Syst, 2015, 26: 1963–1978 -

11 Yu N, Kaneko Y, Ueda M. Supervised Learning Using Spike-Timing-Dependent Plasticity of Memristive Synapses. IEEE Trans Neural Netw Learn Syst , 2015, 26: 2999–3008

12 Lee J H, Delbruck T, Pfeiffer M. Training deep spiking neural networks using backpropagatio. Front Neurosci, 2016, 10: 508

13 Diehl P U, Neil D, Binas J, et al. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In: Proceedings of the International Joint Conference on Neural Networks, 2015: 1–8

14 Hussain S, Liu S-C and Basu A. Improved margin multi-class classification using dendritic neurons with morphological learning. In: Proceedings of the 2014 IEEE International Symposium on Circuits and Systems, 2014: 2640–2643

15 O'Connor P, Neil D, Liu S C, et al. Real-time classification and sensor fusion with a spiking deep belief network. Front Neurosci, 2013, 7: 178

16 Zhang T L, Zeng Y, Zhao D C, et al. A Plasticity-centric Approach to Train the Non-differential Spiking Neural Networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018), New Orleans, USA, 2018: 620–627