

# Memristor-based neural networks with weight simultaneous perturbation training

Chunhua Wang  · Lin Xiong · Jingru Sun · Wei Yao

Received: 24 May 2018 / Accepted: 11 December 2018  
© Springer Nature B.V. 2018

**Abstract** The training of neural networks involves numerous operations on the weight matrix. If neural networks are implemented in hardware, all weights will be updated in parallel. However, neural networks based on CMOS technology face many challenges in the updating phase of weights. For example, derivation of the activation function and error back propagation make it difficult to be realized at the circuit level, even though the back propagation algorithm is rather efficient and popular in neural networks. In this paper, a novel synaptic unit based on double identical memristors is designed, on the basis of which a new neural network circuit architecture is proposed. The whole network is trained by a hardware-friendly weight simultaneous perturbation (WSP) algorithm. The hardware implementation of neural networks based on WSP algorithm only involves the feedforward circuit and does not require the bidirectional circuit. Furthermore, two forward calculations are merely needed to update all weight matrices for each pattern, which significantly simplifies the weight update circuit and allows sim-

pler and easier implementation of the neural network in hardware. The practicability, utility and simplicity of this scheme are demonstrated by the supervised learning tasks.

**Keywords** Memristor · Synaptic unit · Hardware implementation · Multilayer neural networks (MNNs) · Weight simultaneous perturbation (WSP)

## 1 Introduction

With the rapid development of artificial intelligence, neural networks are attracting more and more attentions and play a significantly important role in the field of image recognition, speech recognition [1] and automatic control, etc. Up to now, most of neural networks have been realized by software, which lacks the inherent parallelism of neural networks. However, circuit operations are inherently parallel and capable of providing high speed computation. Therefore, it is necessary to research hardware implementations of neural networks.

Regarding hardware implementations of neural networks, many studies have already been conducted [2–8], most of which implemented neural networks via CMOS technology. However, the implementation of nonvolatile weight storage is a major bottleneck for them. On the other hand, when the area and consumption taken into account, it is a hard task to efficiently

---

C. Wang (✉) · L. Xiong · J. Sun · W. Yao  
College of Computer Science and Electronic Engineering,  
Hunan University, Changsha 410082, China  
e-mail: wch1227164@hnu.edu.cn

L. Xiong  
e-mail: xionglinrurrection@163.com

J. Sun  
e-mail: jt\_sunjr@hnu.edu.cn

W. Yao  
e-mail: yaowei520026@sina.com

realize the circuit-based neural networks using CMOS technology.

Recently, a novel device, the memristor [9–13], provides a fire-new approach for hardware implementations of neural networks. The memristor, a nonvolatile programmable resistor, was first physically realized by HP Labs in 2008 [14]. Current through the memristor or voltage across the memristor enables to tune memristor resistance. When its excitation is turned off, the device keeps its most recent resistance until the excitation is turned on again. Memristor is widely applied to chaotic circuits [15, 16], memory and neural networks. In memristor-based neural networks, memristor's nonvolatility is significantly similar to the biological synapse, which spurs it to be a promising candidate for weight storage. And the fact that memristor belongs to nanodevices enables it easier to be integrated in the analog circuit. Because of these, memristor was widely used in hardware implementations of neural networks.

Many memristor-based neural networks were previously trained by the so-called spike-timing-dependent plasticity (STDP) [17–21], which was intended for explaining the biological phenomenon about tuning synaptic junction strength of the neurons. Besides, Sheri et al. realized neuromorphic character recognition system with two PCMO memristors as a synapse [22]. Nishitani et al. modified an existing three-terminal ferroelectric memristor (3T-FeMEM) model, and the STDP was exploited to perform supervised learning [19]. However, the convergence of STDP-based learning is not guaranteed for general inputs [23].

Other memristor-based learning systems recently attract people's attentions. For example, memristor-based single-layer neural networks (SNNs) trained by the perceptron algorithm were designed to complete some linear separable tasks [24, 25]. Yet, it is the single-layer structure and binary outputs that limit its wide-ranging applications. Therefore, constructing memristor-based multilayer neural networks (MNNs) [26–29] that are rather more practical and efficient than SNNs is requisite. In 2012, Adhikari et al. put forward a modified chip-in-the-loop learning rule for MNNs with memristor bridge synapses that consisted of three transistors and four identical memristors and realized zero, negative and positive synaptic weights [30]. Unlike the conventional chip-in-the-loop learning, a complex multilayer network learning was conducted by decomposing it into multiple simple single-layer network learning, and the gradient descent algorithm was used to train

all single-layer networks [30]. Their design makes the training process simplified and indeed reduces the communication overhead and the circuit complexity. However, their initial training and the weight update calculation were performed in software. In 2015, Soudry et al. designed an artificial synapse constituting of a single memristor and two transistors, and BP algorithm was used to train the whole network [31]. Indeed, it is fairly novel for their design that the errors are encoded as corresponding weight update time and the errors and inputs are introduced into weight updating process by means of the state variables of memristors. However, BP algorithm needs to propagate the error backward from the output layer to the hidden layers, and the update weight is obtained by calculating the derivative of the activation function. These complex calculations are not easy to implement at the circuit level.

Focusing on the difficulties in efficiently implementing learning rule like BP algorithm in hardware, some hardware-friendly algorithms were proposed and applied to the circuit implementation of neural networks. For example, in 2015, the random weight change (RWC) algorithm was used to train the memristor-based neural network where the synaptic weights were randomly updated by a small constant in each iteration [32]. Their design simplifies the circuit structure and does not involve complex circuit operations. However, in RWC's weight updating process, just the sign but not the value of the error variation is considered, and the direction of weight change is random every time, which results in more iterations being needed.

In order to solve the problems in the memristor-based neural network trained by RWC and BP algorithm, in this paper, a new memristor-based neural network circuit trained by weight simultaneous perturbation (WSP) algorithm is proposed. WSP algorithm was introduced in [33]. Perturbations whose signs are randomly selected with equal probability are simultaneously added to all weights, and the difference between the first error function without perturbations and the second one with perturbations is used to control the change of weights in WSP. Unlike RWC algorithm, both the sign and value of the variation of the error function are exploited to update all weights. Hence, fewer iterations are needed compared with RWC algorithm. Additionally, in WSP, complex calculations such as derivation and error back propagation are not also involved compared with BP algorithm. Though the

algorithm is relatively efficient and simple, so far, the memristor-based neural network circuit trained by WSP algorithm is not available. The hardware architecture of the neural network trained by WSP algorithm is therefore proposed in this paper. In addition, in order to solve the problem of storing and updating weights, a novel synaptic unit based on double identical memristors is designed, and taking full advantage of the characteristics of WSP algorithm, this design only needs two forward computations for each pattern to update all weight matrices, which greatly simplifies the circuit structure. This paper indicates that it is possible for neural networks to be actualized in a simpler, more compact and reliable form at the circuit level.

The remainder of this paper is organized as follows. In Sect. 2, the basic information about memristor, RWC algorithm and WSP algorithm is introduced. In Sect. 3, the circuit implementations of SNNs and MNNs are described in detail. In Sect. 4, the proposed circuit architectures are numerically evaluated by supervised tasks to demonstrate its practicability. Finally, Sect. 6 presents conclusions.

## 2 Preliminaries

In this section, the basic information about memristor, RWC algorithm and WSP algorithm is described.

### 2.1 Memristor

In 1971, Chua et al. firstly proposed the missing fourth fundamental circuit element, the memristor, which is a nonlinear passive two-terminal electrical component relating electric charge and magnetic flux linkage [11]. And the memristor’s electrical resistance can be tuned according to its current or voltage. Its physical model [34] is represented by the following formulas:

$$i(t) = G(s(t))v(t) \tag{1}$$

$$\frac{ds(t)}{dt} = v(t) \tag{2}$$

where  $G(s(t))$  is the memductance.  $v(t)$  and  $i(t)$  represent voltage and current, respectively.  $s(t)$  denotes the state variable.

Assume that  $G(s(t))$  is  $n$ -order differentiable near  $s^*$ .  $G(s(t))$  is expanded to the Taylor series at  $s = s^*$  as follows:

$$G(s) = \frac{G(s^*)}{0!} + \frac{G'(s^*)}{1!}(s - s^*) + \frac{G''(s^*)}{2!}(s - s^*)^2 + \dots + \frac{G^{(n)}(s^*)}{n!}(s - s^*)^n + O[(s - s^*)^n] \tag{3}$$

where  $n$  is a positive integer.  $G^{(n)}(s^*)$  represents  $n$ th order derivative of the function  $G(s)$  at  $s = s^*$ , and  $O[(s - s^*)^n]$  represents higher-order infinitesimal of  $(s - s^*)^n$ .

Soudry et al. demonstrated that if the variations in the value of  $s(t)$  are restricted to be small,  $G(s(t))$  is able to be approximated by first-order Taylor series around certain point  $s^*$  [31]. The memductance is hence given by

$$G(s) = \frac{G(s^*)}{0!} + \frac{G'(s^*)}{1!}(s - s^*) \tag{4}$$

simplified form:

$$G(s(t)) = g^* + \hat{g}s(t) \tag{5}$$

where  $\hat{g} = [dG(s)/ds]_{s=s^*}$  and  $g^* = G(s^*) - \hat{g}s^*$ .

Based on this memristor model, SSNs and MNNs are efficiently implemented in hardware.

### 2.2 Random weight change

In the random weight change (RWC) algorithm, synaptic weights are randomly tuned by a small constant, and the learning rule is given as

$$\begin{cases} \omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij}(n+1) \\ \Delta\omega_{ij}(n+1) = \Delta\omega_{ij} & \text{if } E(n+1) < E(n) \\ \Delta\omega_{ij}(n+1) = \delta * Rand(n) & \text{if } E(n+1) \geq E(n) \end{cases} \tag{6}$$

where  $E(\cdot)$  represents the squared error.  $\delta$  is a small constant and  $Rand(n)$  represents the random function which has merely two values, namely  $\pm 1$ . Weights of the network are changed randomly from the initial state with a small increment of  $+\delta$  or  $-\delta$ . If  $E(\cdot)$  decreases by the weight change, the same weight change is iterated until the error increases. If  $E(\cdot)$  increases, weights are updated randomly again [32]. The RWC algorithm is similar to the Brownian weight movement [35], both of which belong to the stochastic learning algorithm [36]. In RWC, only the sign of the variation of the error function is considered, but its value is not fully utilized. Since the absolute value of the weight change is constant in RWC, this value is generally small in each weight update in order to ensure convergence.

Moreover, the direction of weight change is random every time. Hence, the general trend of the trajectory of the weight change is to change along the direction of energy decline, as shown in [32, Fig. 1]. Therefore, more iterations are needed in RWC.

The WSP algorithm is different from the RWC algorithm. In WSP, the sign and value of the variation of the error function are used to tune the weights, which makes the data fully utilized, as described in Sect. 2.3. It also does not require complicated operations such as derivative and error back propagation.

### 2.3 Weight simultaneous perturbation

For convenience, the following nomenclatures are used in this paper.

$\omega_{ij}$  ( $i = 1, 2, \dots, m, j = 1, 2, \dots, n$ ): A weight.

$\Delta\omega_{ij}$  ( $i = 1, 2, \dots, m, j = 1, 2, \dots, n$ ): Variation of  $\omega_{ij}$ .

$h_{ij}$  ( $i = 1, 2, \dots, m, j = 1, 2, \dots, n$ ): A sign.  $h_{ij} = \pm 1$ . The sign of  $h_{ij}$  is randomly selected with equal probability.

$\mathbf{W}$ : A  $m \times n$  weight matrix consisting of  $\omega_{ij}$ .

$\Delta\mathbf{W}$ : A  $m \times n$  weight update matrix consisting of  $\Delta\omega_{ij}$ .

$\mathbf{H}$ : A  $m \times n$  sign matrix consisting of  $h_{ij}$ .

$\mathbf{x}$ : A vector representing a pattern.

$\mathbf{d}$ : A vector representing a target output.

$\mathbf{o}$ : A vector representing an actual output of the network without perturbations for the input  $\mathbf{x}$ .

$\mathbf{o}_{\text{per}}$ : A vector representing an actual output of the network with simultaneous perturbations for the input  $\mathbf{x}$ .

Hence, the relationship between  $\mathbf{x}$  and  $\mathbf{o}$  is given by

$$\mathbf{o} = f(\mathbf{W}\mathbf{x}) \quad (7)$$

where  $f(\cdot)$  represents the activation function. The error function is defined as

$$E = \frac{1}{2} \|\mathbf{d} - \mathbf{o}\|^2 \quad (8)$$

In BP algorithm, the weight update learning rule is

$$\begin{cases} \Delta\omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} \\ \omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij} \end{cases} \quad (9)$$

Regarding the WSP algorithm, its execution details are as follows. Firstly, the normal error function, without simultaneous perturbations, is calculated. Then, perturbations whose signs are randomly selected are simultaneously applied to all weights, and the error

function calculation process is again executed. Finally, the weight update matrices are calculated according to the difference between the first and second error functions. In 1993, Alspector et al. [33] demonstrated that in WSP algorithm, the amount of change in weight was able to be approximated by the following formulas for a perturbation small enough

$$\begin{cases} \Delta\omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} \\ \approx -\eta \frac{E(\omega_{ij} + \omega_{\text{per}}h_{ij}) - E(\omega_{ij})}{\omega_{\text{per}}} h_{ij} \\ \omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij} \end{cases} \quad (10)$$

The learning rule is expressed in matrix form

$$\begin{cases} \Delta\mathbf{W} = -\eta \frac{E(\mathbf{W} + \omega_{\text{per}}\mathbf{H}) - E(\mathbf{W})}{\omega_{\text{per}}} \mathbf{H} \\ = -\eta \frac{\Delta E}{\omega_{\text{per}}} \mathbf{H} \\ \mathbf{W}(n+1) = \mathbf{W}(n) + \Delta\mathbf{W} \end{cases} \quad (11)$$

where  $\omega_{\text{per}}$ , a positive (usually small) constant, is the value of simultaneous perturbation injected into the weight.  $\eta$  is the learning rate, a positive coefficient.

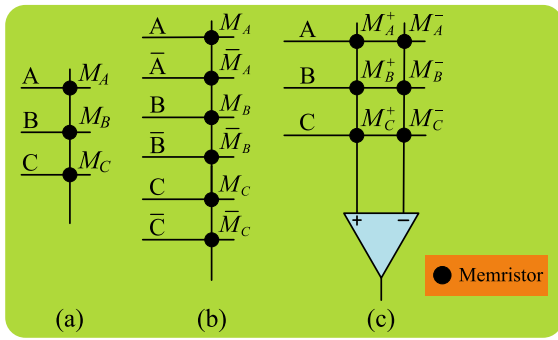
According to Eq. (11), the WSP algorithm requires only two feedforward operations of the neural network. And compared with BP algorithm, derivative of the activation function and error back propagation which are difficult to be implemented in hardware are not involved. In addition to simple sign circuits to control the direction of weight perturbation and weight adjustment, it either does not require other multiplication and transposition operations.

## 3 Circuit architecture design

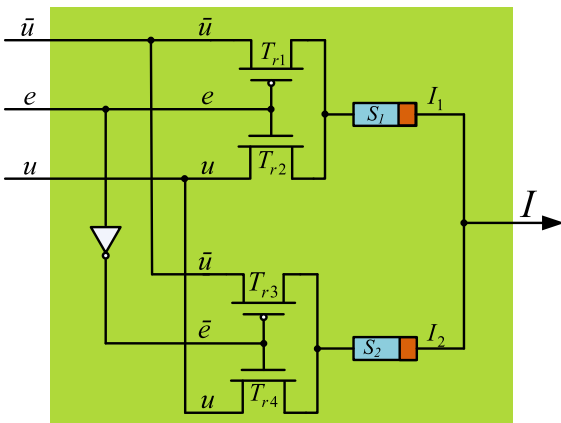
In this section, implementing simple SNNs and general MNNs architectures based on memristor arrays with the WSP algorithm is described. MNNs are generally constructed by cascading multiple SNNs. Therefore, attentions are mainly paid to the design of SNNs.

### 3.1 Artificial synapse

As regards the circuit architecture of the artificial synapse in the memristor crossbars, three possible approaches are exhibited in Fig. 1. In Fig. 1a, a single memristor at the cross point of the circuit is regarded as a synapse for weight storage where the adjustable



**Fig. 1** Three circuit architectures of memristor-based artificial synapse



**Fig. 2** A synaptic unit based on double identical memristors

range of each weight is relatively small. Accordingly, in order to expand this range, the architectures in Fig. 1b, c are extensively employed in memristor crossbars. In Fig. 1b, each input signal and its opposite polarity signal are fed to a column of memristors, and in Fig. 1c each input signal is applied to a pair of memristors [27]. By adding simple auxiliary circuits, two memristors are considered as a synapse. In this paper, the architecture in Fig. 1b is selected.

The architecture constituting of two MOSFET (p-type and n-type) and an ideal single memristor was proposed by Soudry [31, Fig. 3(a)]. Taking full advantage of their architecture, a novel artificial synapse based on double identical memristors is designed, as shown in Fig. 2.

The current of the n-type transistor in the linear region is

$$i_n = k_n \left[ (v_{gs} - v_{th})v_{ds} - \frac{1}{2}v_{ds}^2 \right] \tag{12}$$

where  $v_{gs}$  and  $v_{ds}$  are the gate–source voltage and the drain–source voltage, respectively.  $v_{th}$  is the threshold voltage.  $k_n$  is the conductance parameter of the n-type transistor. When  $v_{gs} < v_{th}$ , the current of the n-type transistor is equal to zero, namely  $i_n = 0$ . Similarly, for the p-type transistor working in the linear region, its current is

$$i_p = -k_p \left[ (v_{gs} - v_{th})v_{ds} - \frac{1}{2}v_{ds}^2 \right] \tag{13}$$

where  $k_p$  is the conductance parameter of the p-type transistor. When  $v_{gs} > -v_{th}$ , the current of the p-type transistor is equal to zero, namely  $i_p = 0$ . Assume that  $k_n = k_p = k$  and the absolute value of the threshold voltage, namely  $|v_{th}|$ , is the same for two transistors.

The synaptic unit is fed with three voltage input signals in which  $u$  and  $\bar{u} = -u$  are, respectively, connected to the terminals of the n-type and p-type transistors and an enable signal  $e$  whose value is 0,  $V_{DD}$ , or  $-V_{DD}$  (with  $V_{DD} > v_{th}$ ) is connected to the gate of both transistors.

We assume that

$$-v_{th} < u < v_{th} \tag{14}$$

According to [31, 37], in order to ensure that the input voltage is mainly applied to the memristor, assume that

$$k(v_{DD} - 2v_{th}) \gg G(s(t)) \tag{15}$$

Consequently, there are three possible situations:

1. When  $e = 0$ , four transistors are off. In other words, the output current of the synaptic unit is equal to zero, namely  $I = 0$ . In this case, the state variables are not changed.
2. When  $e = V_{DD}$ ,  $T_{r2}$  and  $T_{r3}$  both work in the linear region, while  $T_{r1}$  and  $T_{r4}$  are off. The currents  $I_1$  and  $I_2$  through both memristors, respectively, are

$$\begin{cases} I_1 = (g^* + \hat{g}s_1)u \\ I_2 = -(g^* + \hat{g}s_2)u \end{cases} \tag{16}$$

The output current  $I$  of the synaptic unit is

$$I = I_1 + I_2 = (s_1 - s_2)\hat{g}u = \Delta s \hat{g}u \tag{17}$$

3. When  $e = -V_{DD}$ ,  $T_{r1}$  and  $T_{r4}$  both work in the linear region, while  $T_{r2}$  and  $T_{r3}$  are off. The currents  $I_1$  and  $I_2$  through both memristors, respectively, are

$$\begin{cases} I_1 = -(g^* + \hat{g}s_1)u \\ I_2 = (g^* + \hat{g}s_2)u \end{cases} \tag{18}$$

The output current  $I$  of the synaptic unit is

$$I = I_1 + I_2 = (s_2 - s_1)\hat{g}u = \Delta s \hat{g}u \tag{19}$$

where for notational simplicity,  $\Delta s$  contains the sign of the change in two state variables.

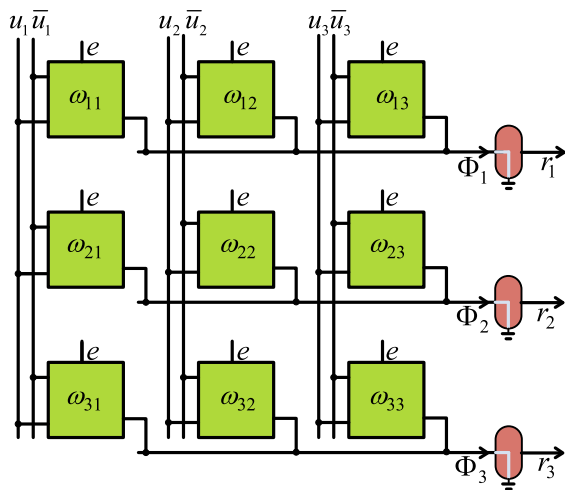
According to the relationship between input signal and output signal from Eqs. (17) and (19), we can define the synaptic weight as

$$\omega = \Delta s \hat{g} \tag{20}$$

The above analysis indicates that the design of the synaptic unit is feasible.

### 3.2 SNN circuit architecture

Based on the proposed synaptic unit in Fig. 2, a  $3 \times 3$  SNN circuit architecture is designed in Fig. 3, where each synaptic unit is indexed by  $(i, j)$ , with  $i \in \{1, \dots, 3\}$  and  $j \in \{1, \dots, 3\}$ . Each synaptic unit receives two inputs  $u_j, \bar{u}_j$  and an enable signal  $e_{ij}$ , and produces an output current  $I_{ij}$ . Each column of synaptic units in the arrays shares two inputs  $u_j$  and  $\bar{u}_j$ . The current summing operation,  $\sum_j I_{ij}$ , is conducted at the terminal of each row. The row output interfaces convert the total current to the dimensionless output signal  $r_i$ .



**Fig. 3** A  $3 \times 3$  SNN circuit architecture.  $\omega_{ij}$  represents a synaptic unit which receives two voltage input signals  $u_j, \bar{u}_j$  and an enable signal  $e_{ij}$ , and produces an output current  $I_{ij}$ , wherein the terminal  $e_{ij}$  controls not only the sign of input signal but also the duration and direction of weight adjustment signal. And the brown box represents the row output interface for data conversion. ( $i \in \{1, \dots, 3\}, j \in \{1, \dots, 3\}$ ). (Color figure online)

### 3.3 Circuit operation

During running period of WSP algorithm, the circuit's operations in each trial are composed of simple five phases. First, in the computing phase, the current summation operation and calculation of the error function are realized. Second, in the perturbing phase, perturbations with random signs are added to all weights simultaneously. Third, a new computing phase is needed in order to get the new error function. Fourth, in the restoring phase, restoring signals are applied to memristors to eliminate effect of the perturbation in the perturbing phase on the state variables. Fifth, in the updating phase, all weight matrices are adjusted according to Eq. (11). The signals and their effect on the state variable are shown in Fig. 4.

(1) *First computing phase* A vector  $\mathbf{x}$  from the dataset is encoded by multiplying a positive constant  $a$ , which allows to convert a dimensionless number  $x_j$  to corresponding voltage  $u_j$ , namely  $\mathbf{u} = a\mathbf{x}$ . In order not to change the state of the memristor at last, when the input signal is greater than zero, namely  $x_j > 0$ , the enable signal is designed as follows:

$$e_{ij}(t) = \begin{cases} V_{DD}, & \text{if } 0 \leq t < 0.5T_1 \\ -V_{DD}, & \text{if } 0.5T_1 \leq t < T_1 \end{cases} \tag{21}$$

When the input signal is less than zero, namely  $x_j < 0$ , the enable signal is designed as follows:

$$e_{ij}(t) = \begin{cases} -V_{DD}, & \text{if } 0 \leq t < 0.5T_1 \\ V_{DD}, & \text{if } 0.5T_1 \leq t < T_1 \end{cases} \tag{22}$$

According to Eq. (2), the total changes of the state variables for both memristors are, respectively, given by

$$\begin{cases} \Delta s_{ij1} = \int_0^{0.5T_1} (ax_j)dt + \int_{0.5T_1}^{T_1} (-ax_j)dt = 0 \\ \Delta s_{ij2} = \int_0^{0.5T_1} (-ax_j)dt + \int_{0.5T_1}^{T_1} (ax_j)dt = 0 \end{cases} \tag{23}$$

From Eq. (23), the net changes of the state variables both are zero, which indicates that the resistance of both memristors is not finally changed. When  $t \in [0, 0.5T_1)$ , the transistors  $T_{r2}$  and  $T_{r3}$  both work in the linear region, while  $T_{r1}$  and  $T_{r4}$  correspondingly are in the cutoff region. The currents  $I_1$  and  $I_2$  in Fig. 2 are calculated according to Eq. (16):

$$\begin{cases} I_{ij1} = a(g^* + \hat{g}s_{ij1})x_j \\ I_{ij2} = -a(g^* + \hat{g}s_{ij2})x_j \end{cases} \tag{24}$$

And the output current  $I$  of each synaptic unit from Eq. (17) is

$$I_{ij} = I_{ij1} + I_{ij2} = a\hat{g}(s_{ij1} - s_{ij2})x = a\hat{g}s_{ij}x \tag{25}$$

Therefore, the total current in each row terminal  $\Phi_i$  is

$$\Phi_i = \sum_j I_{ij} = a\hat{g} \sum_j s_{ij}x_j \tag{26}$$

The target output vector  $\mathbf{d}$  from the dataset commonly is dimensionless. The constant  $c$ , consequently, is significantly essential to convert the current unit to a dimensionless number in the row output interface, namely

$$o_i = c\Phi_i = ac\hat{g} \sum_j s_{ij}x_j \tag{27}$$

According to Eq. (20), we obtain

$$\omega_{ij} = ac\hat{g}s_{ij} \tag{28}$$

On the basis of the Ohm's law, the arithmetic operation  $\mathbf{W}\mathbf{x}$  is realized in this circuit. Finally, according to the target output  $\mathbf{d}$  and the actual output  $\mathbf{o}$ , the first error function also called cost function in machine learning is calculated and stored in a buffer circuit. The error function is given by

$$E_{\text{fir}} = \frac{1}{2} \|\mathbf{d} - \mathbf{o}\|^2 \tag{29}$$

(2) *Perturbing phase* During perturbing phase, a pair of voltage signals  $u_{\text{per}}$  and  $\bar{u}_{\text{per}}$  with unchanging amplitude and duration are applied to the terminals  $u$  and  $\bar{u}$  in order to alter the state of memristors and introduce the simultaneous perturbation. To generate random perturbation, let  $e = h_{ij}V_{\text{DD}}$  control the sign of each perturbation, as described in Section III-A and the duration of  $\bar{u}_{\text{per}}$  is the same as that of  $e$ .

$$e_{ij} = h_{ij}V_{\text{DD}} \quad \text{if } T_1 \leq t < T_2 \tag{30}$$

From Eq. (28), when  $t \in [T_1, T_2)$ , the perturbation constant  $\omega_{\text{per}}$  is given by

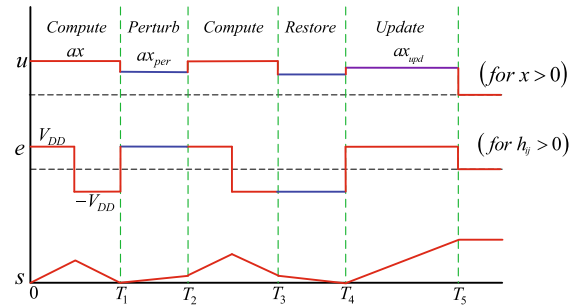
$$s_{\text{per}} = \frac{\omega_{\text{per}}}{ac\hat{g}} \tag{31}$$

From the definitional perspective of the state variable, the total change in the internal state for the synaptic unit is

$$\begin{cases} u_{\text{per}} = ax_{\text{per}} \\ \Delta s_{\text{per}} = \Delta s_1 - \Delta s_2 \\ \quad = \int_{T_1}^{T_2} [u_{\text{per}} - (-u_{\text{per}})] dt \\ \quad = 2ax_{\text{per}}(T_2 - T_1) \end{cases} \quad \text{if } T_1 \leq t < T_2 \tag{32}$$

where the absolute values of  $\Delta s_1$  and  $\Delta s_2$  are equal because two memristors of the synaptic unit are identical. According to Eqs. (31) and (32), the duration of the perturbing signal with a constant amplitude is

$$T_{\text{per}} = T_2 - T_1 = \left| \frac{\omega_{\text{per}}}{2a^2c\hat{g}x_{\text{per}}} \right| \tag{33}$$



**Fig. 4** Operation protocol of a single synaptic unit (note that for the curve of  $S$  in the figure, we intentionally enlarge the variation of  $S$  in the perturbing phase and the restoring phase, and expand the duration of the perturbing phase and the restoring phase in order to better observe its change)

where  $T_{\text{per}}$  is a positive constant. In other words, the duration of the perturbing phase is changeless throughout the training process, which also explains why the magnitude and duration of the voltage signals  $u_{\text{per}}$  and  $\bar{u}_{\text{per}}$  are fixed.

(3) *Second computing phase* During each computing phase, the operations of the entire network are the same as the first computing phase. In this phase, the enable signal is ( for  $x_j > 0$ )

$$e_{ij}(t) = \begin{cases} V_{\text{DD}}, & \text{if } T_2 \leq t < \frac{T_2+T_3}{2} \\ -V_{\text{DD}}, & \text{if } \frac{T_2+T_3}{2} \leq t < T_3 \end{cases} \tag{34}$$

And the input signals are the same as in the first computing phase. Hence, the total changes in the internal state variables for the synaptic unit are

$$\begin{cases} \Delta s_{ij1} = \int_{T_2}^{\frac{T_2+T_3}{2}} (ax_j)dt + \int_{\frac{T_2+T_3}{2}}^{T_3} (-ax_j)dt \\ \quad = 0 \\ \Delta s_{ij2} = \int_{T_2}^{\frac{T_2+T_3}{2}} (-ax_j)dt + \int_{\frac{T_2+T_3}{2}}^{T_3} (ax_j)dt \\ \quad = 0 \end{cases} \tag{35}$$

And the error function is given by

$$E_{\text{sec}} = \frac{1}{2} \|\mathbf{d} - \mathbf{o}_{\text{per}}\|^2 \tag{36}$$

(4) *Restoring phase* During the perturbing phase, simultaneous perturbations are added to change the state of memristors. Restoring signals are, hence, needed to restore memristors to the stage where simultaneous perturbations are not added, which is significantly critical for accurate update of all weights. Yet, there are some differences compared with the perturbing phase. The enable signal and input signal are, respectively,

$$\begin{cases} u_{\text{per}} = ax_{\text{per}} \\ e_{ij} = -h_{ij}V_{\text{DD}} \end{cases} \text{ if } T_3 \leq t < T_4 \quad (37)$$

And the duration of  $e$  and  $u$  must meet the following requirement:

$$T_{\text{per}} = T_4 - T_3 \quad (38)$$

(5) *Updating phase* During updating phase, the updating voltage signals with a constant amplitude, namely  $u_{\text{upd}}$  and  $\bar{u}_{\text{upd}}$ , are, respectively, connected to the terminals  $u$  and  $\bar{u}$ . And the difference between two error functions is given by

$$\Delta E = E(\mathbf{W} + \omega_{\text{per}}\mathbf{H}) - E(\mathbf{W}) = E_{\text{sec}} - E_{\text{fir}} \quad (39)$$

According to the learning rule described by Eq. (11), the variation of each weight is

$$\Delta\omega_{ij} = -\eta \frac{\Delta E}{\omega_{\text{per}}} h_{ij} \quad (40)$$

From Eq. (28), we obtain

$$\Delta s_{ij} = -\eta \frac{\Delta E}{ac\hat{g}\omega_{\text{per}}} h_{ij} \quad (41)$$

The direction of weight adjustment is controlled by the enable signal  $e$ .

$$\begin{cases} e_{ij} = V_{\text{DD}} & \text{if } \Delta\omega_{ij} > 0 \\ e_{ij} = 0 & \text{if } \Delta\omega_{ij} = 0 \\ e_{ij} = -V_{\text{DD}} & \text{if } \Delta\omega_{ij} < 0 \end{cases} \quad (42)$$

From Eq. (2), the variation of the state variable for the synaptic unit is, hence, (for  $\Delta\omega_{ij} > 0$ )

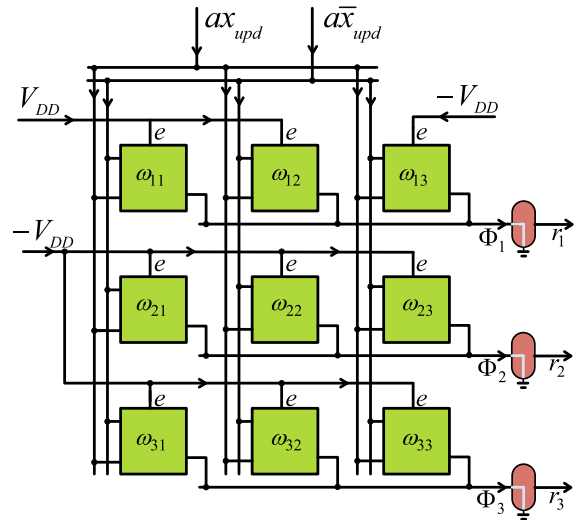
$$\begin{cases} u_{\text{upd}} = ax_{\text{upd}} \\ \Delta s_{ij} = \int_{T_4}^{T_5} [u_{\text{upd}} - (-u_{\text{upd}})] dt \\ = \int_{T_4}^{T_5} 2ax_{\text{upd}} dt \\ = 2a(T_5 - T_4)x_{\text{upd}} \end{cases} \text{ if } T_4 \leq t < T_5 \quad (43)$$

According to Eq. (41) and Eq. (43), the duration of the updating signal with a constant amplitude  $ax_{\text{upd}}$  is

$$T_{\text{upd}} = T_5 - T_4 = \left| \eta \frac{\Delta E}{2a^2c\hat{g}\omega_{\text{per}}x_{\text{upd}}} \right| \quad (44)$$

Therefore, the sign and duration of the input and update signals can be easily controlled by the enable signal  $e$ . Finally, for example, assume that the signs of  $\Delta\omega_{11}$  and  $\Delta\omega_{12}$  are plus and others are minus in Fig. 3. So, its weight update circuit is designed as Fig. 5.

Through the above analysis, there are three noteworthy points:



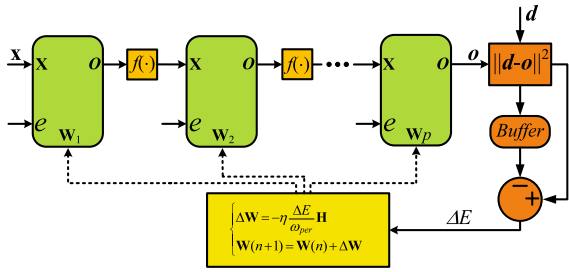
**Fig. 5** Circuit structure of weight updating when  $\Delta\omega_{11}$  and  $\Delta\omega_{12}$  both are positive and others are negative. The duration of the enable signal is  $T_{\text{upd}}$

1. In the whole training process, the duration and amplitude of the perturbation signal, namely  $T_{\text{per}}$  and  $ax_{\text{per}}$ , are constants, and the random sign of the perturbation signal is controlled by the terminal  $e$ .
2. The duration of computing and restoring phase for each pattern is also constant throughout the training process.
3. In the  $k$ th weight update process, the duration and amplitude of the updating signal, namely  $T_{\text{upd}}$  and  $ax_{\text{upd}}$ , are constants for all synaptic units, and the sign of the updating signal is controlled by the terminal  $e$ . ( $k$ , a positive integer, is the index of weight update.)

### 3.4 MNN circuit architecture

In part 2 and part 3, attentions are primarily paid to the SNNs circuit architecture and its operations with the WSP algorithm training. And derivative of the activation function and error back propagation are not involved for the multilayer neural network in WSP, which greatly simplifies the circuit structure. Therefore, in this part, the MNNs circuit architecture cascading multiple SNNs, mainly a two-layer neural network, is discussed. Regarding the circuit architecture of MNNs, its schematic is described in Fig. 6.





**Fig. 6** Architecture of a general MNN trained by WSP algorithm.  $f(\cdot)$  represents the activation function. The green box represents a layer in a multilayer network. (Color figure online)

For a two-layer neural network, its mathematical model is expressed as

$$\mathbf{o} = \mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x}) \tag{45}$$

where  $f(\cdot)$  denotes the activation function.  $\mathbf{W}_1$  and  $\mathbf{W}_2$  are the input-hidden and the hidden-output weight matrix, respectively. And from Eq. (11), the weight update rule for a two-layer network is given by

$$\begin{cases} \Delta \mathbf{W}_1 = -\eta \frac{\Delta E}{\omega_{per}} \mathbf{H}_1 \\ \Delta \mathbf{W}_2 = -\eta \frac{\Delta E}{\omega_{per}} \mathbf{H}_2 \end{cases} \tag{46}$$

where  $\mathbf{H}_1$  and  $\mathbf{H}_2$  are, respectively, the random sign matrices of  $\mathbf{W}_1$  and  $\mathbf{W}_2$ . The learning rule for a general MNNs is derived based on Eq. (11)

$$\Delta \mathbf{W}_p = -\eta \frac{\Delta E}{\omega_{per}} \mathbf{H}_p \tag{47}$$

where the subscript  $p$ , a positive integer, is the index of network layer without taking the input layer into account.

As for the circuit operations of MNNs, it is basically the same as that of SNNs. Similarly, it is only two feed-forward operations that are needed to synchronously update all weight matrices for the MNN when WSP algorithm used to train the neural network according to Eq. (47).

### 4 Circuit simulation

In this section, the neural network circuit is modeled in SimElectronics attached to the Simscape toolbox in MATLAB to evaluate the circuit performance. The memristor model [31] proposed by Soudry is exploited

**Table 1** Circuit parameters

Parameter	Value
$u_{per}$	40 mV
$a$	0.1 V
$V_{DD}$	5 V
$T_1$	20 $\mu$ s
$c$	$1.0 \times 10^8$ A
$g^*$	1 $\mu$ S
$\hat{g}$	180 $\mu$ S V <sup>-1</sup> s <sup>-1</sup>

in this paper. In SimElectronics, basic electrical component models including the MOSFET are provided. In addition, other circuit operations required for realizing the WSP algorithm such as random sign generation with equal probability and calculation of the error function are implemented using MATLAB code. As for the weight updating, a voltage signal with a constant magnitude, namely  $ax_{upd}$  in Fig. 4, is needed, and its polarity is controlled by the enable signal  $e$ . Hence, the weight update value is converted to corresponding pulse of specific duration from Eq. (44). Other parameters in this paper are shown in Table 1.

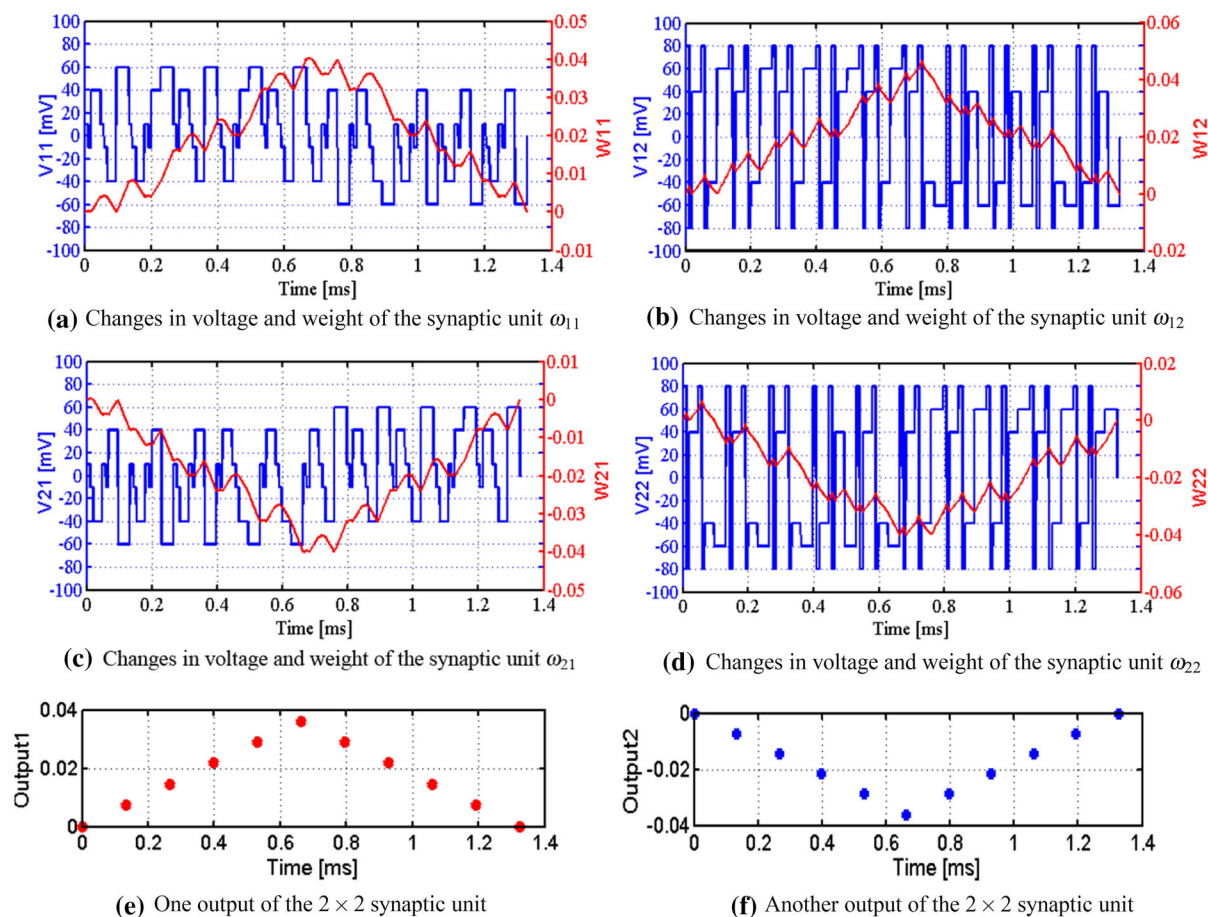
#### 4.1 Basic operations of synaptic unit

To verify basic operations of the proposed synaptic unit, a numerical simulation is carried out. A relatively simple  $2 \times 2$  synaptic unit circuit whose input signals  $x_1$  and  $x_2$  are separately set to 0.1 and 0.8 is simulated in ten operation cycles, namely  $t \in [0, 10T_5]$ . The simultaneous perturbation  $\omega_{per}$  is 0.004, and its sign is randomly chosen with equal probability. The magnitude of updating signal, namely  $ax_{upd}$ , is 60 mV. With regard to weight adjustments of four synapse units, assume that

$$\begin{cases} \Delta \omega_{11} = 0.008 \times \text{sign}(5T_5 - t) \\ \Delta \omega_{12} = 0.008 \times \text{sign}(5T_5 - t) \\ \Delta \omega_{21} = 0.008 \times \text{sign}(t - 5T_5) \\ \Delta \omega_{22} = 0.008 \times \text{sign}(t - 5T_5) \end{cases} \tag{48}$$

It is because the synaptic unit is a dual identical memristors structure that the absolute value of the corresponding weight change for each memristor is 0.004.

Based on the above hypothetical data, the circuit simulation results are shown in Fig. 7, together with the input voltages of synaptic units and its correspond-



**Fig. 7** A  $2 \times 2$  synaptic unit circuit simulation during ten operation cycles. Top and middle: voltage (solid blue) and weight (solid red) change for each synaptic unit. Bottom: corresponding

output result (Output1, Output2).  $v_{ij}$  is the input signal connected to the terminal  $u$  of the synaptic unit ( $i \in [1, 2], j \in [1, 2]$ ). (Color figure online)

ing weights. The feasibility of basic operations of the circuit is finally verified.

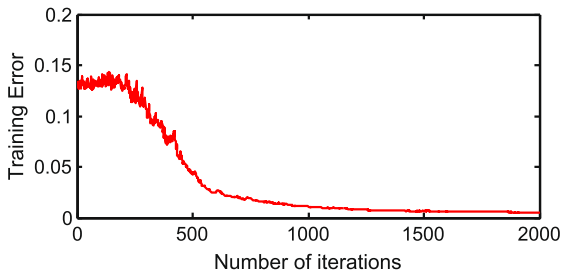
#### 4.2 Three input odd parity function

A three input odd parity function task is conducted using a two-layer network with five neurons in the hidden layer and only one neuron in the output layer. Table 2 shows the truth table of the three odd parity function. In this task, the activation function is set as the sigmoid function. The learning rate  $\eta$  is 0.2, and the significantly important parameter  $\omega_{\text{per}}$ , the amount of weight simultaneous perturbation, is set to 0.002. If  $\omega_{\text{per}}$  is too large, the whole system may not converge from Eq. (11). On the contrary, if it is too small, the

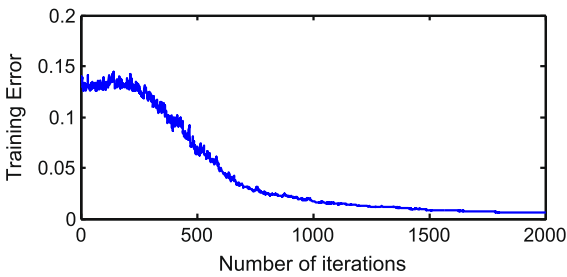
number of iterations will become very large. The magnitude of perturbation voltage, namely  $ax_{\text{per}}$ , is 40 mV. The curve between network training errors and iterations is shown in Fig. 8. Eight different test samples, all possible inputs of three input odd parity function, are finally used to test the performance of the network. The target output and actual output are shown in Fig. 9 where the mean squared errors (MSE) are 0.0016. As shown in Fig. 8, the iterations trained with WSP algorithm are about 1000. For the same task, the iterations trained with RWC algorithm are approximately 10,000 and the iterations of BP are about 1000 in [38, Fig. 12], which indicates that the performance of the proposed memristor-based neural network with WSP training is successful.

**Table 2** Truth table of three input parity function

$x_1$	$x_2$	$x_3$	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



(a)

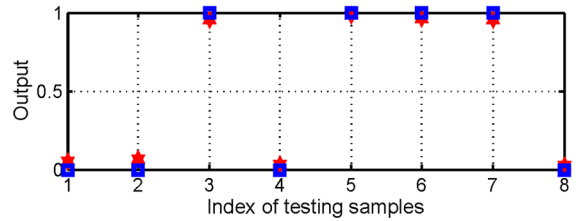


(b)

**Fig. 8** Training error versus iterations curve. **a** The training error with algorithm. **b** The training error with circuit

### 4.3 Iris classification

Another more complex classification task, Iris classification, is conducted to test accuracy of the proposed memristor-based architecture trained by WSP algorithm. The Iris dataset [39] contains 150 patterns which can be divided into three classes (Setosa, Versicolor, Virginica), and each pattern consists of four features which are normalized in this task. The simulation is executed using the designed two-layer network with four neurons in the hidden layer and three neurons in the output layer. In this task, the activation function is set as the sigmoid function. The three



**Fig. 9** The target output and the actual output of circuit simulation for eight samples. The red and blue dots represent the actual output and the target output, respectively. (Color figure online)

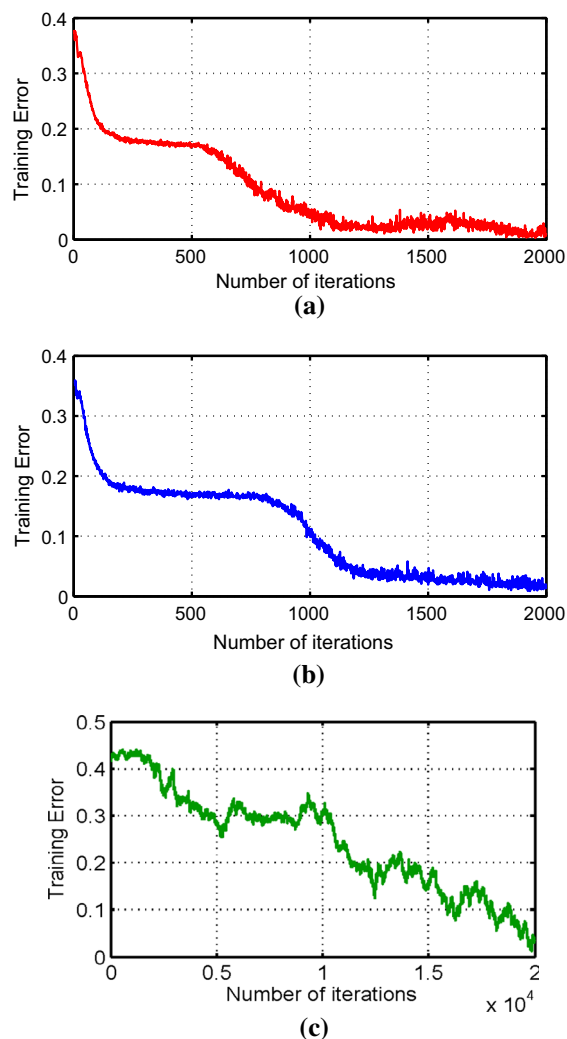
classes of the dataset are encoded with three bits, and the details are shown in Table 3 in which Output-X1, Output-X2 and Output-X3 represent outputs of three neurons in the output layer, respectively. In terms of network parameters, the learning rate  $\eta$  is 0.02, and the significantly important parameter  $\omega_{per}$  is set to 0.001. The training set size is 120. Finally, the training error is shown in Fig. 10, and 30 test samples are selected to test the network performance. In Fig. 11, the target output and actual output are depicted. The MSE of each bit in Fig. 11a–c are 0.0004, 0.0012 and 0.0006, respectively. The iterations trained with RWC algorithm are approximately 20,000 in Fig. 10c, while that of WSP are approximately 2,000 in Fig. 10b. The iterations of BP are about 1200 in [31, Fig. 8(b)]. Compared with BP algorithm, the iterations with WSP algorithm training are more. But comparing [31, Fig.4(b)] with Fig. 6, the circuit structure trained by WSP algorithm is simpler, because it does not involve complicated circuit operations, such as derivative of the activation function and error back propagation. These results indicate that the proposed scheme is practical.

### 5 Discussion

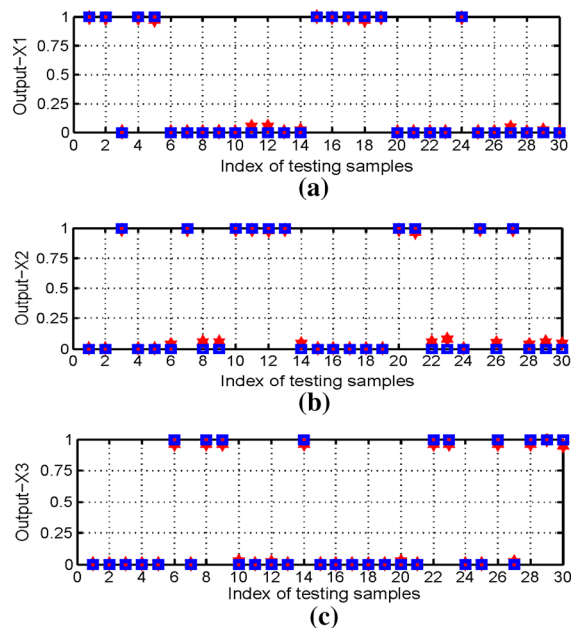
The iterations of WSP, RWC and BP algorithms are shown in Table 4 for three input odd parity function and Iris classification. WSP algorithm is an approximation to the gradient descent algorithm, so its convergence speed is smaller than that of the BP algorithm based on Eq. (10). RWC algorithm is based on the Brownian random motion equation, so the convergence speed is smaller than WSP algorithm based on Eq. (6). Although converging faster than RWC and WSP, BP algorithm involves complex operations such as the derivative of activation function and error back propagation. And in BP, the weight changes depend on gradient of the error

**Table 3** Coding list for three classes

Category	Encoded output		
	Output-X1	Output-X2	Output-X3
Setosa	1	0	0
Versicolor	0	1	0
Virginica	0	0	1

**Fig. 10** Training error versus iterations curve. **a** The training error with algorithm. **b** The training error with circuit. **c** The training error with WRC algorithm

function. When the error function has a great number of local minima, the neural network easily converges to local optimum. Many proposed solutions use a perturbation inserted in the network in order to get out from local minima. The introduction of noise during

**Fig. 11** The target output and the actual output of circuit simulation for 30 test samples. **a–c**, respectively, are results of the output neurons Output-X1, Output-X2 and Output-X3. The red and blue dots represent the actual output and the target output, respectively. (Color figure online)

learning allows the solution to reach the global optimum. Furthermore, it has been shown that training with noise may improve the generalization capability of the network and reduce the negative effects of the weight variations after learning [40]. The advantages of WSP with respect to the BP are the ability to avoid local minima of the error function and the low computational complexity, which leads to a simplification in hardware implementation.

## 6 Conclusion

A memristor-based hardware implementation of the neural network with on-chip learning is proposed.

**Table 4** The iterations of RWC, BP and WSP algorithms

Algorithm	Task	
	Three input odd parity function	Iris classification
RWC	10,000	20,000
BP	900 [40]	1200 [40]
WSP	1000	2000

Moreover, WSP algorithm, a hardware-friendly algorithm, is adopted to train the neural network, which does not require error back propagation and calculation of the derivative of activation function. Only two feed-forward operations are needed to update all weights. Therefore, the circuit architecture is simpler and more compact compared with BP algorithm. By combining nonvolatile memristors and WSP algorithm, a simple, compact and reliable hardware implementations of neural networks is put in real-life practice. Two supervised learning tasks, three input odd parity function and Iris classification, are conducted to test the accuracy of the network, and the correctness of the proposed architecture is similar to its software counterpart, which opens an opportunity for large-scale online learning in MNNs.

**Acknowledgements** This work was supported in part by the National Natural Science Foundation of China (No. 61571185), the Natural Science Foundation of Hunan Province, China (No. 2016JJ2030) and the Open Fund Project of Key Laboratory in Hunan Universities (No. 15K027).

#### Compliance with ethical standards

**Conflict of interest** The authors declare that they have no conflict of interest.

#### References

- Tao, F., Busso, C.: Gating neural network for large vocabulary audiovisual speech recognition. *IEEE/ACM Trans. Audio Speech Lang. Process. (TASLP)* **26**(7), 1286–1298 (2018)
- Talaška, T., Kolasa, M., Dlugosz, R., Pedrycz, W.: Analog programmable distance calculation circuit for winner takes all neural network realized in the CMOS technology. *IEEE Trans. Neural Netw. Learn. Syst.* **27**(3), 661–673 (2016)
- Talaška, T., Kolasa, M., Dlugosz, R., Farine, P.A.: An efficient initialization mechanism of neurons for Winner Takes All Neural Network implemented in the CMOS technology. *Appl. Math. Comput.* **267**, 119–138 (2015)
- Sengupta, A., Banerjee, A., Roy, K.: Hybrid spintronic-CMOS spiking neural network with on-chip learning: devices, circuits, and systems. *Phys. Rev. Appl.* **6**(6), 064003 (2016)
- Dlugosz, R., Kolasa, M., Pedrycz, W., Szulc, M.: Parallel programmable asynchronous neighborhood mechanism for Kohonen SOM implemented in CMOS technology. *IEEE Trans. Neural Netw. Learn. Syst.* **22**(12), 2091–2104 (2011)
- Wu, X., Saxena, V., Zhu, K., Balagopal, S.: A CMOS spiking neuron for brain-inspired neural networks with resistive synapses and in situ learning. *IEEE Trans. Circuits Syst. II* **62**(11), 1088–1092 (2015)
- Pan, C., Naeemi, A.: Non-Boolean computing benchmarking for beyond-CMOS devices based on cellular neural network. *IEEE J. Explor. Solid-State Comput. Devices Circuits* **2**, 36–43 (2016)
- Goknar, I.C., Yildiz, M., Minaei, S., Deniz, E.: Neural CMOS-integrated circuit and its application to data classification. *IEEE Trans. Neural Netw. Learn. Syst.* **23**(5), 717–724 (2012)
- Valov, I., Kozicki, M.: Non-volatile memories: organic memristors come of age. *Nat. Mater.* **16**, 1170–1172 (2017)
- Wang, Z., Joshi, S., Savel'ev, S.E., Jiang, H., Midya, R., Lin, P., Wu, Q.: Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nat. Mater.* **16**(1), 101 (2017)
- Herrmann, E., Rush, A., Bailey, T., Jha, R.: Gate controlled three-terminal metal oxide memristor. *IEEE Electron Device Lett.* **39**(4), 500–503 (2018)
- van de Burgt, Y., Lubberman, E., Fuller, E.J., Keene, S.T., Faria, G.C., Agarwal, S., Salleo, A.: A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing. *Nat. Mater.* **16**(4), 414–418 (2017)
- Gupta, I., Serb, A., Khiat, A., Zeitler, R., Vassanelli, S., Prodromakis, T.: Sub 100 nW volatile nano-metal-oxide memristor as synaptic-like encoder of neuronal spikes. *IEEE Trans. Biomed. Circuits Syst.* **12**(2), 351–359 (2018)
- Strukov, D.B., Snider, G.S., Stewart, D.R., Williams, R.S.: The missing memristor found. *Nature* **453**(7191), 80 (2008)
- Zhou, L., Wang, C., Zhou, L.: Generating hyperchaotic multi-wing attractor in a 4D memristive circuit. *Nonlinear Dyn.* **85**(4), 2653–2663 (2016)
- Zhou, L., Wang, C., Zhou, L.: A novel no-equilibrium hyperchaotic multi-wing system via introducing memristor. *Int. J. Circuit Theory Appl.* **46**(1), 84–98 (2018)
- Panwar, N., Rajendran, B., Ganguly, U.: Arbitrary spike time dependent plasticity (STDP) in memristor by analog waveform engineering. *IEEE Electron Device Letters.* **38**(6), 740–743 (2017)
- Cai, W., Ellinger, F., Tetzlaff, R.: Neuronal synapse as a memristor: modeling pair- and triplet-based STDP rule. *IEEE Trans. Biomed. Circuits Syst.* **9**(1), 87–95 (2015)
- Nishitani, Y., Kaneko, Y., Ueda, M.: Supervised learning using spike-timing-dependent plasticity of memristive synapses. *IEEE Trans. Neural Netw. Learn. Syst.* **26**(12), 2999–3008 (2015)
- Boyn, S., Grollier, J., Lecerf, G., Xu, B., Locatelli, N., Fusil, S., et al.: Learning through ferroelectric domain dynamics in solid-state synapses. *Nat. Commun.* **8**, 14736 (2017)

21. Kheradpisheh, S.R., Ganjtabesh, M., Thorpe, S.J., Masquelier, T.: STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* **99**, 56–67 (2018)
22. Sheri, A.M., Hwang, H., Jeon, M., Lee, B.G.: Neuromorphic character recognition system with two PCMO memristors as a synapse. *IEEE Trans. Ind. Electron.* **61**(6), 2933–2941 (2014)
23. Legenstein, R., Naeger, C., Maass, W.: What can a neuron learn with spike-timing-dependent plasticity? *Neural Comput.* **17**(11), 2337–2382 (2005)
24. Alibart, F., Zamanidoost, E., Strukov, D.B.: Pattern classification by memristive crossbar circuits using ex situ and in situ training. *Nat. Commun.* **4**, 2072 (2013)
25. Prezioso, M., Merrih-Bayat, F., Hoskins, B.D., Adam, G.C., Likharev, K.K., Strukov, D.B.: Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* **521**, 61–64 (2015)
26. Li, C., Belkin, D., Li, Y., Yan, P., Hu, M., Ge, N., Song, W., et al.: Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nat. Commun.* **9**, 2385 (2018)
27. Bayat, F.M., Prezioso, M., Chakrabarti, B., Nili, H., Kataeva, I., Strukov, D.: Implementation of multilayer perceptron network with highly uniform passive memristive crossbar circuits. *Nat. Commun.* **9**, 2331 (2018)
28. Hu, X., Feng, G., Duan, S., Liu, L.: A memristive multilayer cellular neural network with applications to image processing. *IEEE Trans. Neural Netw. Learn. Syst.* **28**(8), 1889–1901 (2016)
29. Zeng, X., Wen, S., Zeng, Z., Huang, T.: Design of memristor-based image convolution calculation in convolutional neural network. *Neural Comput. Appl.* **30**(2), 503–508 (2018)
30. Adhikari, S.P., Yang, C., Kim, H., Chua, L.O.: Memristor bridge synapse-based neural network and its learning. *IEEE Trans. Neural Netw. Learn. Syst.* **23**(9), 1426–1435 (2012)
31. Soudry, D., Di Castro, D., Gal, A., Kolodny, A., Kvatinsky, S.: Memristor-based multilayer neural networks with online gradient descent training. *IEEE Trans. Neural Netw. Learn. Syst.* **26**(10), 2408–2421 (2015)
32. Adhikari, S.P., Kim, H., Budhathoki, R.K., Yang, C., Chua, L.O.: A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses. *IEEE Trans. Circuits Syst. I* **62**(1), 215–223 (2015)
33. Alspector, J., Meir, R., Yuhua, B., Jayakumar, A., Lippe, D.: A parallel gradient descent method for learning in analog VLSI neural networks. In: *Advances in Neural Information Processing Systems*, pp. 836–844 (1993)
34. Kvatinsky, S., Friedman, E.G., Kolodny, A., Weiser, U.C.: Team: threshold adaptive memristor model. *IEEE Trans. Circuits Syst. I* **60**(1), 211–221 (2013)
35. Belli, M.R., Conti, M., Turchetti, C.: Analog Brownian weight movement for learning of artificial neural networks. In: *European Symposium on Artificial Neural Networks (ESANN)*, pp. 19–21 (1995)
36. Conti, M., Orcioni, S., Turchetti, C.: A new stochastic learning algorithm for analog hardware implementation. In: *International Conference on Artificial Neural Networks (ICANN)*, pp. 1171–1176 (1998)
37. Kvatinsky, S., Satat, G., Wald, N., Friedman, E.G., Kolodny, A., Weiser, U.C.: Memristor-based material implication (imply) logic: design principles and methodologies. *IEEE Trans. Very Large Scale Integr. Syst.* **22**(10), 2054–2066 (2014)
38. Yang, C., Kim, H., Adhikari, S.P., Chua, L.O.: A circuit-based neural network with hybrid learning of backpropagation and random weight change algorithms. *Sensors* **17**, 16 (2016)
39. Bache, K., Lichman, M.: UCI machine learning repository. <http://archive.ics.uci.edu/ml> (2018)
40. Murray, A.F., Edwards, P.J.: Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Trans. Neural Netw.* **5**(5), 792–802 (1994)