Complex & Intelligent Systems (2022) 8:787–802
https://doi.org/10.1007/s40747-021-00282-4

ORIGINAL ARTICLE

Memristor crossbar architectures for implementing deep neural networks

Xiaoyang Liu[1,2] · Zhigang Zeng[1,2]

Received: 13 October 2020 / Accepted: 20 January 2021 / Published online: 20 July 2021

Abstract

The paper presents memristor crossbar architectures for implementing layers in deep neural networks, including the fully connected layer, the convolutional layer, and the pooling layer. The crossbars achieve positive and negative weight values and approximately realize various nonlinear activation functions. Then the layers constructed by the crossbars are adopted to build the memristor-based multi-layer neural network (MMNN) and the memristor-based convolutional neural network (MCNN). Two kinds of in-situ weight update schemes, which are the fixed-voltage update and the approximately linear update, respectively, are used to train the networks. Consider variations resulted from the inherent characteristics of memristors and the errors of programming voltages, the robustness of MMNN and MCNN to these variations is analyzed. The simulation results on standard datasets show that deep neural networks (DNNs) built by the memristor crossbars work satisfactorily in pattern recognition tasks and have certain robustness to memristor variations.

Keywords Memristor · Memristor-based neural network · Deep neural network · Multi-layer neural network · Convolutional neural network · Neuromorphic architecture

Introduction

Great progress has been made in DNNs in recent years. DNNs have excellent performance in image recognition, speech recognition, machine translation and related fields and have been widely used in artificial intelligence. But as tasks become more and more complex, the requirement of computing power becomes higher and higher. The traditional computing devices based on von Neumann architecture suffer the memory wall problem due to the separation of the computing units and the storage units, which hinders the further improvement of their computing capability. Memristors achieve in-memory and parallel computing, accelerating the operation of DNNs. Meanwhile the plasticity of memristor is very similar to that of synapse. Memristors also have advantages of low power consumption and nanoscale, and are compatible with the complementary metal-oxide-semiconductor (CMOS) technology. Therefore memristors are promising elements to build new computing architectures.

The existence of memristor was predicted in theory in 1971 [6]. Since the memristor was first manufactured in 2008 [31], the research on memristor-based neural networks has developed rapidly. Memristors laid out in the form of crossbars have low power consumption, high density, and could perform the vector-multiplication in parallel. There have been various memristor-based neural networks, such as single-layer and multi-layer neural networks (SNNs, MNNs) [3,4,7,10–12,19,27,30,41,42,45,49,50], convolutional neural networks (CNNs) [8,21,26,36,39,40,46,49], Pavlov associative memory networks [5,24,25,34,43,52], long short-term memory networks (LSTMs) [1,2,9,20,23,28,29,35], pulse coupled neural networks (PCNNs) [38,53], hierarchical temporal memory (HTM) [13,14,22,33,54].

A memristor bridge synapse-based neural network is proposed in [3]. The memristor bridge synapse achieves positive or negative synaptic weight value using four memristors. A modified chip-in-the-loop learning scheme is put forward to train the network. In [4], a memristor-based SNN

✉ Zhigang Zeng
zgzeng@hust.edu.cn

Xiaoyang Liu
xyliu@hust.edu.cn

1 Key Laboratory of Image Processing and Intelligent Control of Education Ministry of China, Wuhan, China

2 School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China

is presented and it is trained with ex-situ and in-situ methods. The signed weight value is achieved by subtracting the conductance value of one memristor from that of another memristor. The results show that memristor-based networks are promising implementation of neuromorphic computing systems. In [10], memristor crossbar-based neural network with on-chip back propagation (BP) training is presented. Memristor-based multi-layer neural networks with online gradient descent training are presented in [30]. The network uses one memristor and two CMOS transistors to construct one synapse. Compared with COMS-based counterparts, the memristor-based MNNs [30] consume between 2% and 8% of the area and static power. In [48], a sign backpropagation (SBP) method is proposed to train the resistive random access memory (RRAM)-based neural networks. In [50], circuit design for memristor-based MNNs is presented and a modified BP algorithm is adopted to train the networks. In [19], the in-situ learning capability of the MNN based on the hafnium oxide-based memristor crossbar is experimentally demonstrated. In [49], memristor-based quantized neural networks are presented. The weights are quantized to accelerate the operation of the neural networks.

Besides memristor-based MNNs, there are also works for memristor-based CNNs. In [39], a memristor-based CNN is presented, which is the first time that the memristor-based circuit implements CNN. One memristor crossbar represents all groups of convolution kernels in one convolutional layer and performs the convolutional operation. An extremely parallel implementation of memristor crossbar-based CNN is presented in [40]. It uses a very sparse crossbar reproducing convolution kernels to implement the convolutional operation, and one feature map is convolved at a time. In [8], convolutional layers are mapped to resistive cross-point arrays, and the impacts of noises and bound limitations on the performance of the CNN are analyzed. In [36], a memristor-based fully convolutional network (MFCN) is put forward for semantic segmentation tasks. A fully hardware-implemented memristor-based CNN is presented in [46]. High-yield, high-performance, and uniform memristor crossbars are reported in [46], and an effective hybrid-training method which could adapt to device imperfections is put forward to train the memristor crossbar-based neural networks.

In this paper, memristor-based crossbar architectures, which have few elements in each synapse circuit and meanwhile approximately achieves many activation functions, for implementing memristor-based DNNs are presented. In the crossbars, signed weight values are achieved by subtracting the conductance values of memristors from that of reference resistors [32]. Nonlinear activation functions are approximately implemented through circuits. MMNN and MCNN are built by the presented crossbars, which also substantiate the effectiveness of the crossbars. The networks are trained by two kinds of in-situ update schemes, which are the fixed-voltage update and the approximately linear update. The performance of MMNN and MCNN trained by the two update schemes is analyzed. The robustness of the two networks to conductance variations, which are caused by the inherent characteristics of memristors and errors of programming voltages, is also explored.

The rest of the paper is organized as follows. Section "Memristor crossbar architectures" introduces the memristor model and the memristor crossbar architectures designed for fully connected (FC) layer, convolutional operation, and average pooling operation. Section "Operation of the memristor-based DNNs" introduces the operation of the DNNs built by the crossbars. Simulations and analyses are conducted in Section "Simulations and analysis". Section "Conclusions" concludes the paper.

Memristor crossbar architectures

In this section, memristor crossbar architectures for DNNs are presented. Memristor crossbars perform vector-matrix multiplications, which are computational complicated operations in neural networks, in parallel through Kirchhoff's law. This section first introduces the memristor model and then presents memristor crossbars for the FC layer, convolutional operation, and average pooling operation. These memristor-based crossbars could be used to build DNNs.

Memristor model

The memristor model is established to describe the behavior of realistic memristor in mathematical formula, and it can be used to explore characteristics of the memristor. It can also be adopted in simulations to speed up the system design. The HP model is [31]

$$v(t) = i(t)R(t), \tag{1}$$
$$R(t) = R_{on}x(t) + R_{off}\left(1 - x(t)\right), \tag{2}$$

where $R(t)$ is the resistance of the memristor, $x(t)$ is the state variable, R_{on} and R_{off} are the internal low and high resistance of the memristor, respectively, and $v(t)$ and $i(t)$ are the voltage and the current, respectively. And

$$x(t) = \frac{w(t)}{D}, \tag{3}$$
$$\frac{\mathrm{d}w(t)}{\mathrm{d}t} = \mu_v \frac{R_{on}}{D} i(t)\, f\left(x(t)\right), \tag{4}$$

where $w(t)$ is the internal state variable, D is the thickness, and μ_v is the average ion mobility. D and μ_v are constants.

The HP model can not model characteristics of many realistic memristors precisely, therefore various memristor models have been put forward to describe behaviors of different memristors [17,18,37,51]. A voltage controlled threshold model [51] that can fit realistic memristors is adopted in the paper

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \begin{cases} \mu_v \frac{R_{on}}{D^2} \frac{i_{off}}{i(t)-i_0} f(x(t)), & 0 < V_{on} < v(t) \\ 0, & V_{on} \le v(t) \le V_{off} \\ \mu_v \frac{R_{on}}{D^2} \frac{i(t)}{i_{on}} f(x(t)), & v(t) < V_{off} < 0 \end{cases} \quad (5)$$

where i_0, i_{on}, and i_{off} are constants, and $f(x(t))$ is the window function which is defined as

$$f(x(t)) = 1 - (2x(t) - 1)^2. \quad (6)$$

Memristor crossbar for FC layer

The FC layer is the basic unit to constitute MNN and is also an essential part of CNN. In the FC layer, inputs are weighted and summed, that is

$$y_j = f\left(\sum_{i=1}^{M} W_{ji} x_i\right), \quad (7)$$

where x_i is the ith input, y_j is the jth output, W_{ji} is the weight value between the ith input unit and the jth output unit, M is the number of input units, and $f(\cdot)$ is the activation function which could be the binary function, the sigmoid function, the rectified linear unit (ReLU), or the hyperbolic tangent (tanh) function.

The memristor crossbar for the FC layer is shown in Fig. 1. Through Kirchhoff's law, the memristor crossbar implements the weighted summing up operation in (7), whose computation complexity is generally $O(N^2)$, with computation complexity of $O(1)$. In the inference phase, TGs (transmission gates) in the left column and the row below the memristor rows are closed and there is

$$V_f = -R_f \left(\sum_{i=1}^{M} \frac{V_{x,i}}{R_s} + \frac{V_b}{R_s}\right). \quad (8)$$

$V_{x,i} = V_r \cdot x_i$, where V_r is the read voltage and x_i is the original input value. The current of the jth column is

$$\begin{aligned} I_j &= \sum_{i=1}^{M} \frac{V_{x,i}}{R_{i,j}} + \frac{V_b}{R_{M+1,j}} + \frac{V_f}{R_f} \\ &= \sum_{i=1}^{M} V_{x,i} \cdot (G_{i,j} - G_s) \\ &\quad + V_b \cdot (G_{M+1,j} - G_s), \end{aligned} \quad (9)$$

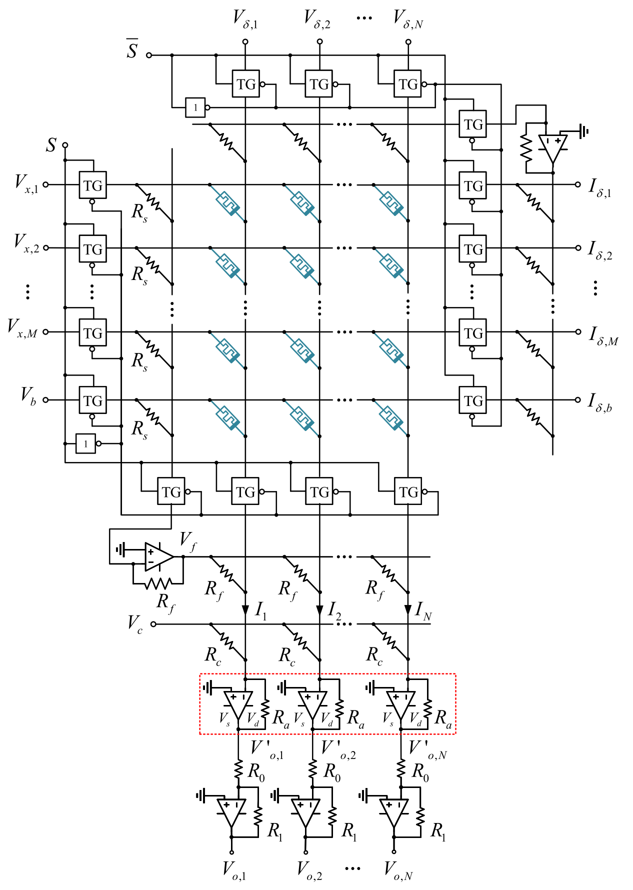

Fig. 1 The memristor crossbar for the FC layer. In the forward pass, $V_{x,1}$ to $V_{x,M}$ are input voltages representing input values of the layer, $V_{o,1}$ to $V_{o,N}$ are output voltages representing output values. In the training phase, $V_{\delta,1}$ to $V_{\delta,N}$ are errors to be back propagated and $I_{\delta,1}$ to $I_{\delta,M}$, $I_{\delta b}$ are back propagated errors. TG is the transmission gate, and S and $\bar{S}$ are signals to switch inference and learning phases

where $G_s = 1/R_s$ and $G_{i,j} = 1/R_{i,j}$ $(j = 1, 2, \ldots, N)$ is the conductance of the memristor in the ith row and the jth column. The output voltage is

$$-V'_{o,j} = I_j R_a + \frac{V_c R_a}{R_c}. \quad (10)$$

Through setting different values of R_a and R_c, various activation functions can be approximately achieved. Denote the source voltages of amplifiers in the dotted box by V_s and V_d. When the resistance of R_a is very large and $V_s = 0$, $V_d = -1$V, then it can be approximately obtained that

$$-V'_{o,j} = \begin{cases} 1, & I_j > 0 \\ 0. & I_j \le 0 \end{cases} \quad (11)$$

It is a binary activation function.

Set $R_a = 0.25\mathrm{V}/\left(V_r r_{gw}\right)$, $V_c R_a / R_c = 0.5\mathrm{V}$, where V is volt, and then

$$-V'_{o,j} = 0.25\frac{I_j}{V_r r_{gw}}\mathrm{V} + 0.5\mathrm{V}, \tag{12}$$

where $I_j/\left(V_r r_{gw}\right)$ is the numerical value of the output of the jth column and r_{gw} is the ratio of the conductance value and the weight value. Let $x = I_j/\left(V_r r_{gw}\right)$, $y = -V'_{o,j}$, $V_s = 0\mathrm{V}$, and $V_d = -1V$. Ignoring the voltage unit volt, there is

$$y = \begin{cases} 1, & x > 2 \\ 0.25x + 0.5, & -2 \le x \le 2 \\ 0. & x < -2 \end{cases} \tag{13}$$

This formula is an approximate realization of the sigmoid function [39].

Similarly, set $R_a = 1\mathrm{V}/\left(V_r r_{gw}\right)$, $V_c = 0$, $V_s = 1\mathrm{V}$, and $V_d = -1\mathrm{V}$, there is

$$y = \begin{cases} 1, & x > 1 \\ x, & -1 \le x \le 1 \\ -1, & x < -1 \end{cases} \tag{14}$$

It approximately achieves the tanh function [35].

Set $R_a = 1\mathrm{V}/\left(V_r r_{gw}\right)$, $V_c = 0$, $V_s = 0$, and $V_d = -v_H$, and then

$$y = \begin{cases} v_H, & x > v_H \\ x, & 0 \le x \le v_H \\ 0. & x < 0 \end{cases} \tag{15}$$

It is an approximate realization of the ReLU function with an upper bound of v_H.

Then

$$V_{o,j} = -\frac{R_1}{R_0}V'_{o,j}, \tag{16}$$

where $\frac{R_1}{R_0}$ is to rescale the amplitude of the output voltage to be within thresholds of memristors.

For the classification layer, the activation function part of the crossbar is shown in Fig. 2, and it meanwhile calculates the error between the prediction and the target which is

$$\Delta V_j = V_{o,j} - V_{T,j}. \tag{17}$$

In the training phase, TGs in the first row and the right column are closed by setting S low level and $\bar{S}$ high level. And now the crossbar back propagates errors. $V_{\delta,1}$ to $V_{\delta,N}$ are errors to be back propagated and $I_{\delta,1}$ to $I_{\delta,M}$, $I_{\delta,b}$ are back propagated errors.

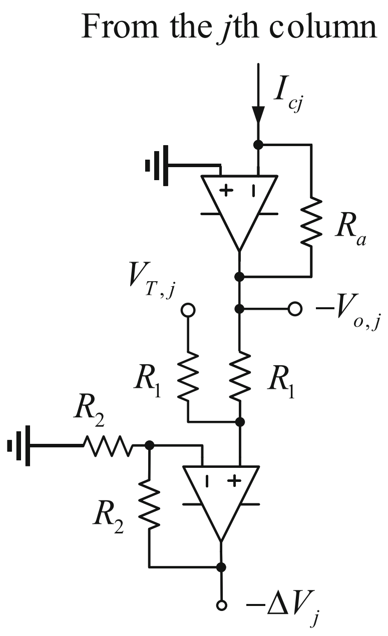

Fig. 2 The activation function part of the classification layer, which also calculates the error. $V_{o,j}$ is the jth activated output and $V_{T,j}$ is the target value for $V_{o,j}$. ΔV_j is the error between $V_{o,j}$ and $V_{T,j}$, and $\Delta V_j = V_{o,j} - V_{T,j}$

Memristor crossbar for convolutional operation

The convolutional operation uses several groups of convolution kernels to convolve feature maps, as shown in Fig. 3. The number of kernel groups is equal to the number of output feature maps. The size of each kernel is $K_1 \times K_2$, where K_1 and K_2 are the width and height of the kernel, respectively. The convolutional operation is

$$y_j^p = \sum_{i=1}^{M}\sum_{k_1=1}^{K_1}\sum_{k_2=1}^{K_2} W_{k_1,k_2,i,j}x^p_{k_1,k_2,i}, \tag{18}$$

where y_j^p $(j = 1, 2, \ldots, N)$ is the pth value in the jth output feature map, $x^p_{k_1,k_2,i}$ is the value at the position (k_1, k_2) of the pth receptive field in the ith input feature map, $W_{k_1,k_2,i,j}$ is the weight value at the position (k_1, k_2) of the ith kernel in the jth kernel groups, M is the number of input channels, and N is the number of output channels. Suppose the convolution stride is s, and the padding size is P, the dimension of the output feature map is

$$\left(N, \left[\left(H_1 - K_1 + 2P\right)/s\right] + 1, \left[\left(H_2 - K_2 + 2P\right)/s\right] + 1\right), \tag{19}$$

where H_1 and H_2 is the width and the height of the input feature map, respectively, and $[\cdot]$ is the integral function.

There are two methods to implement the convolutional operation by means of memristor crossbar. One is to consider

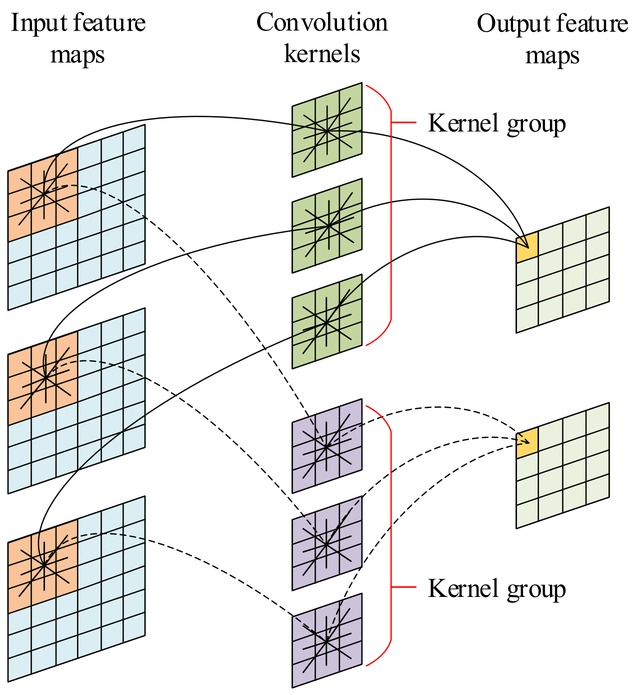

Fig. 3 The convolutional operation. There are three input feature maps, two output feature maps, and two groups of convolution kernels. Each kernel group contains three kernels and these kernels convolve three input feature maps and generate one output feature map

a compact memristor crossbar as a set of sliding windows that slide over input feature maps in turn to obtain the output feature map [39]. The other is to input an entire feature map to a sparse crossbar [40], but this method needs lots of redundant memristors and it is also challenging to make the conductance of the same convolution kernel the same. This paper adopts the first method whose crossbar scale is much smaller.

The memristor crossbar for convolutional operation is shown in Fig. 4. The current of the jth column is

$$I_j^p = \sum_{i=1}^{M} \sum_{k=1}^{K_1\times K_2} \frac{V_{k,i}^p}{R_{k,i,j}} + \frac{V_b}{R_{M\times K_1\times K_2+1,j}} + \frac{V_f}{R_f}$$
$$= \sum_{i=1}^{M} \sum_{k=1}^{K_1\times K_2} V_{k,i}^p \cdot \left(G_{k,i,j} - G_s\right) + V_b \cdot \left(G_{M\times K_1\times K_2+1,j} - G_s\right), \tag{20}$$

where $G_{k,i,j}$ is the conductance value of the memristor in the jth column that receives $V_{k,i}^p$ and $G_{M\times K_1\times K_2+1,j}$ is the conductance of the memristor in the jth column that receives V_b. Then

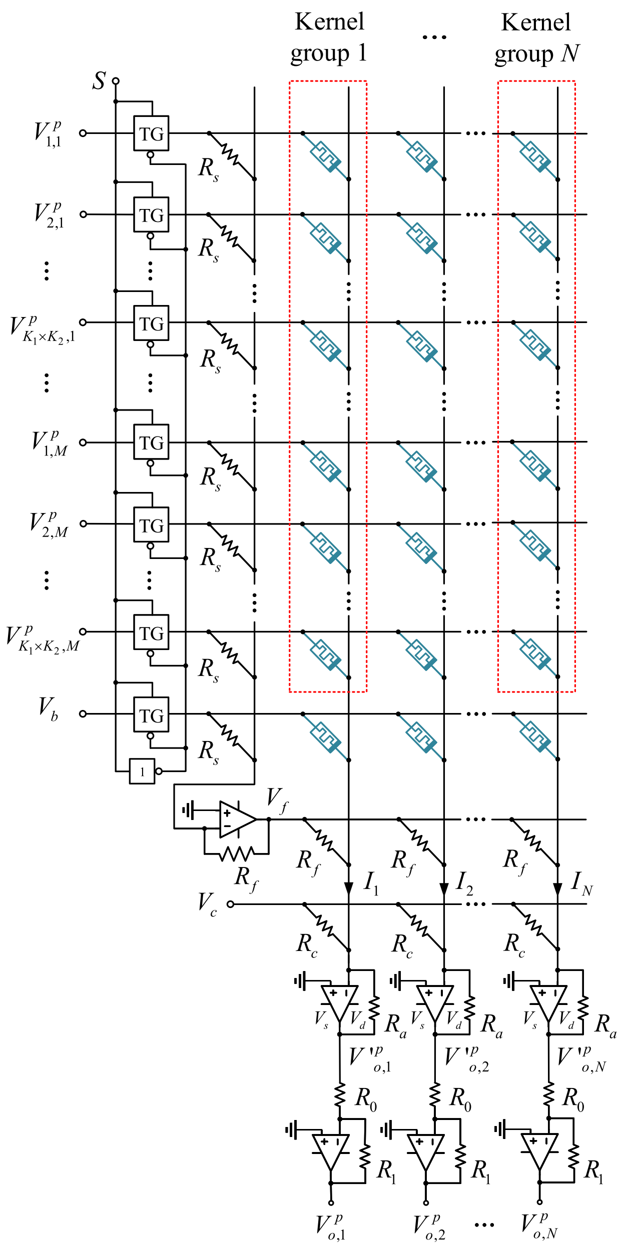

Fig. 4 The memristor crossbar for convolutional operation. It is considered as sliding windows. Each column contains M convolution kernels, and the number of columns N is the same as the number of output channels. $V_{k,i}^p$ ($k = 1, 2, \ldots, K_1 \times K_2,\ i = 1, 2, \ldots, M,$) is the kth input value from the pth receptive field in the ith input feature map, and K_1 and K_2 are the width and height of the kernel, respectively. $V_{o,j}^p$ ($j = 1, 2, \ldots, N$) represents the pth value of the jth output feature map

$$V_{o,j}^p = \frac{R_1}{R_0}\left(I_j^p R_a + \frac{V_c R_a}{R_c}\right), \tag{21}$$

where j is also the index of the jth output feature map. Each column in the crossbar represents one kernel group.

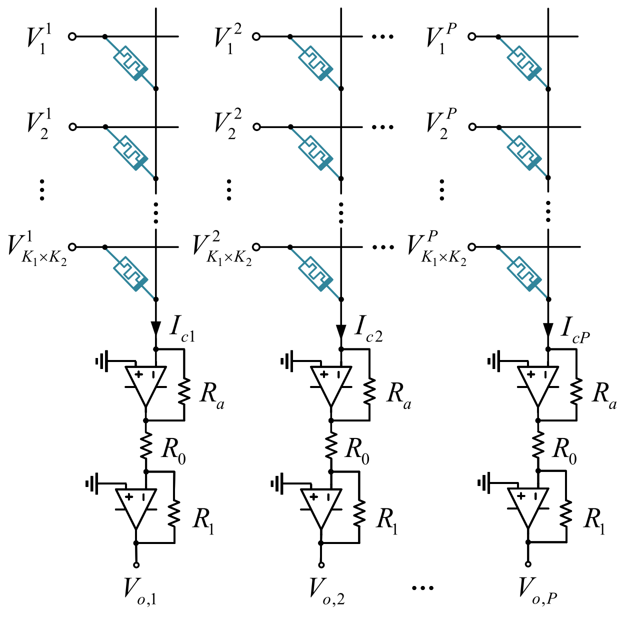

Fig. 5 The memristor array for average pooling operation. $K_1 \times K_2$ is the pooling kernel size and superscript p ($p = 1, 2, \ldots, P$) indicates the pth pooling region

Memristor array for average pooling operation

The average pooling operation is

$$y^p = \sum_{i=1}^{K_1}\sum_{j=1}^{K_2} \frac{x_{ij}^p}{K_1 \times K_2}, \tag{22}$$

where x_{ij}^p is the input value at the position (i, j) of the pth receptive field, y^p is the output value of the pth receptive field, and K_1 and K_2 is the width and the height of the pooling kernel, respectively. This operation could be implemented by convolutional operation whose stride size is equal to the kernel size and all weight values are $1/(K_1 \times K_2)$.

The memristor array for average pooling operation is shown in Fig. 5. All memristors have resistance values of $K_1 \times K_2 \times R_a$, where R_a represents the resistance of the resistor R_a in Fig. 5. The output voltage of each column is

$$V_{o,p} = \frac{\mathrm{R}_a\mathrm{R}_1}{\mathrm{R}_0} \sum_{i=1}^{K_1\times K_2} \frac{V_i^p}{K_1 \times K_2}, \tag{23}$$

where $p = 1, 2, \ldots, P$.

Operation of the memristor-based DNNs

The memristor-based DNNs are trained through the error back propagation (BP) algorithm. The memristors are updated in-situ according to the weight update value. The advantages of in-situ learning is that the learning process can adjust to hardware imperfections [4,19,47], and the memristors can be updated in parallel. The in-situ learning also provides a possible solution for completely on-chip learning.

Weight update schemes

Two kinds of weight update schemes are adopted to in-situ update memristors in the crossbar. They are the fixed-voltage update and the approximately linear update.

Fixed-voltage update

The fixed-voltage update means that the amplitudes and the duration of writing voltages are fixed. There are two kinds of writing voltages, which are the voltage to increase the conductance and the voltage to decrease the conductance, respectively, and they are different in sign and duration. Which one of them is used depends on the sign of the weight update value. This method is very easy to implement because there is no need to precisely convert weight update values to appropriate writing voltages which is a difficult process because of the nonlinearity of the conductance changing of the memristor [47,48]. If $\Delta W \geq \sigma$, the corresponding memristor is applied the positive writing voltage, and if $\Delta W < -\sigma$, the memristor is applied the negative writing voltage, where ΔW is the weight update value and σ is a small non-negative constant to filter small update values. Because of the nonlinearity, the conductance updating values of all memristors are not the same. Denote the absolute values of the rising slope and the descending slope of the approximately linear region of conductance changing versus timing of writing voltages by k_r and k_d, respectively. The ratio of the duration of the pulse to increase the conductance and that of the pulse to decrease the conductance is equal to k_d/k_r.

Approximately linear update

The approximately linear update means that the middle approximately linear region of conductance changing of the memristor is adopted to represent most weight values [23]. The desired conductance update value $\Delta G = \Delta W \cdot r_{gw}$. The desired voltage duration for adjusting the memristor is approximately calculated as $\Delta G/k_r$ for increasing or $\Delta G/k_d$ for decreasing the conductance.

Update memristors in the crossbar

Memristors are updated by applying voltages with appropriate durations through a row-parallel updating method. Illustrate the method by Fig. 6 in which the conductance of M_1 and M_3 needs to be increased and that of M_2 needs

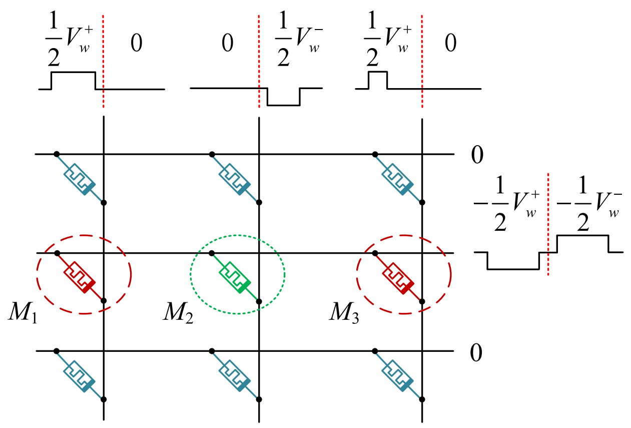

Fig. 6 The row-parallel updating method. $V_w^+ > V_{on} > \frac{1}{2}V_w^+ > 0$ and $V_w^- < V_{off} < \frac{1}{2}V_w^- < 0$, and voltages are divided into two phases. Memristors in the second row are to be updated. Amplitudes of voltages across the first and the third memristors in the second row are both V_w^+, so their conductance is increased in the first phases of column voltages. The amplitude of the voltage across the second memristor in the second row is V_w^-, so its conductance is decreased in the second phase. The conductance of other memristors remains unchanged because their voltages do not exceed threshold voltages. The amount of the change of conductance is determined by the duration of the column voltage

to be decreased. The voltages are divided into two phases. The first phase is to increase the conductance and the second phase is to decrease the conductance. V_w^+ and V_w^- satisfy that $V_w^+ > V_{on} > \frac{1}{2}V_w^+ > 0$ and $V_w^- < V_{off} < \frac{1}{2}V_w^- < 0$. In the first phase, the amplitude of the row voltage is $-\frac{1}{2}V_w^+$ column voltages for increasing the conductance are all $\frac{1}{2}V_w^+$. So only voltages across M_1 and M_3 are beyond the positive threshold voltage of the memristor. In the second phase, the amplitude of the row voltage is $-\frac{1}{2}V_w^-$ and amplitudes of column voltages for decreasing the conductance are all $\frac{1}{2}V_w^-$. So only the voltage across M_2 is below the negative threshold voltage. Therefore only memristors in the second row are updated and the rest remain unchanged. For the fixed-voltage update, pulse durations of columns voltages do not vary, but for the approximately linear update, pulse durations are related to weight update values.

The BP training

The BP training of the network is completed through following steps

1. Reset all memristors to R_{off} by applying reset voltages V_w^-, and then adjust the conductance to the approximately linear region by V_w^+ with appropriate timing.
2. S is set high level and TGs in the left column in the memristor crossbar are closed. Feed input voltages to the DNNs and obtain errors through (17). Then the loss is calculated as

$$\mathcal{L} = \frac{1}{2}\sum_{j=1}^{C} |V_{o,j} - V_{T,j}|^2, \tag{24}$$

where C is the number of classes. Or

$$\mathcal{L} = \frac{1}{2}\|\mathbf{V}_o - \mathbf{V}_T\|_2^2, \tag{25}$$

where $\mathbf{V}_o$ is the final output voltage vector and $\mathbf{V}_T$ is the target voltage vector.
3. Back propagate errors from the $(l+1)$th layer to the lth layer through weights of the $(l+1)$th layer.
For the FC layer, the error voltage vector of the lth layer is

$$\Delta\mathbf{V}^{(l)} = \begin{cases} (\mathbf{V}_o - \mathbf{V}_T) \odot f_l'(\mathbf{V}_z), & \text{if } l = L \\ \left(\left(\mathbf{W}^{(l+1)}\right)^{\mathrm{T}} \Delta\mathbf{V}^{(l+1)}\right) \odot f_l'\left(\mathbf{V}_z^{(l)}\right), & l < L \end{cases} \tag{26}$$

where $\mathbf{W}^{(l+1)}$ is the weight matrix of the $(l+1)$th layer, L is the number of layers, $f_l'(\cdot)$ is the derivation of the activation function in the lth layer, $\mathbf{V}_z^{(l)}$ is the unactivated output voltage vector, and $\odot$ is the element-wise multiplication. The backpropagation can be implemented through the memristor crossbar with S being low level, and now the columns are fed error voltages and the rows output propagated values.
For the convolution layer, there is

$$\Delta\mathbf{V}^{(l)} = \Delta\mathbf{V}^{(l+1)} \otimes \mathbf{rot180}\left(\mathbf{W}^{(l+1)}\right) \odot f_l'\left(\mathbf{V}_z^{(l)}\right), \tag{27}$$

where $\mathbf{W}^{(l+1)}$ is one kernel in the $(l+1)$th layer, $\Delta\mathbf{V}^{(l+1)}$ is the corresponding receptive field in the error matrix, $\otimes$ is the convolution operation, and $\mathbf{rot180}(\cdot)$ is the function to rotate the matrix 180 degrees. The backpropagation is implemented through weights read out from the crossbar.
4. Determine the weight update values. For the FC layer

$$\Delta\mathbf{W}^{(l)} = \Delta\mathbf{V}^{(l)}\left(\mathbf{V}_o^{(l-1)}\right)^{\mathrm{T}}, \tag{28}$$

where $\mathbf{V}_o^{(l-1)}$ is the output voltage vector of the $(l-1)$th layer.
For the convolution layer

$$\Delta\mathbf{W}^{(l)} = \mathbf{V}_o^{(l-1)} \otimes \Delta\mathbf{V}^{(l)}. \tag{29}$$

Table 1 Simulation parameters

Parameter	Value	Parameter	Value
R_{on} (kΩ)	10	D (nm)	1
R_{off} (kΩ)	100	V_w^+ (V)	1.8
R_s (kΩ)	20	V_w^- (V)	−1.8
r_{gw} (S)	3.33×10^{-5}	V_{on} (V)	1.4
i_{on} (A)	12	V_{off} (V)	−1.4
i_{off} (A)	3×10^{-10}	V_r (V)	1.0
i_0 (A)	6×10^{-7}	k_r	2.90
μ_v ($\mathrm{m^2 s^{-1} \Omega^{-1}}$)	1×10^{-12}	k_d	−7.04
Approximately linear region	$[3 \times 10^{-5}, 7 \times 10^{-5}]$	t^+ (ns)	22
σ	0	t^- (ns)	10

5. Determine desired writing voltages. For the fixed-voltage update, the pulse durations of writing voltages are

$$\mathbf{t}_{inc} = (\Delta \mathbf{W} \geq \sigma) \cdot t_0^+, \quad (30)$$
$$\mathbf{t}_{dec} = (\Delta \mathbf{W} < -\sigma) \cdot t_0^-, \quad (31)$$

where t_0^+ and t_0^- are pulse durations of writing voltages for increasing and decreasing conductance, respectively, and amplitudes of these two voltages are V_w^+ and V_w^-, respectively.
For the approximately linear update, conductance update values of memristors are

$$\Delta \mathbf{G} = \Delta \mathbf{W} \cdot r_{gw}. \quad (32)$$

Then desired pulse durations of writing voltages are

$$\mathbf{t}_{inc} = \left[\Delta \mathbf{G} \geq \left(r_{gw} \cdot \sigma\right)\right] / k_r, \quad (33)$$
$$\mathbf{t}_{dec} = \left[\Delta \mathbf{G} < \left(-r_{gw} \cdot \sigma\right)\right] / k_d. \quad (34)$$

6. S is set low level. Apply desired writing voltages to memristors to update their conductance through the introduced weight update schemes.
7. Repeat Step 2 to Step 6 until the loss is smaller than a predefined threshold value.

Because the main purpose of the paper is to evaluate the performance of the memristor-based DNNs constructed by the presented memristor crossbars along with the weight update scheme, the input and the intermediate data of the convolution operation are processed and stored in peripheral digital circuit, update values and so is the calculation of the desired conductance update values and durations of writing voltages. The BP process can also be achieved by analog circuit [15] and the conductance update value can also be determined by look up table (LUT) [49].

Table 2 The truth table of the XOR operation

$in1$	$in2$	out
0	0	0
1	0	1
0	1	1
1	1	0

Simulations and analysis

MMNN and MCNN are built in this section by the presented memristor crossbars to carry out simulation experiments. The effectiveness of the circuits is substantiated in SPICE. The circuits and the learning process are also evaluated in Matlab under hardware defined constraints. The parameters of simulations are listed in Table 1. In the forward pass, the activation functions are the pseudo formulas (13), (14), (15), and in the backward pass, they are based on their original formulas.

Results of MMNN

Two-layer neural networks are built based on the memristor-based crossbar in Fig. 1 for XOR operation and digits recognition on MNIST (Modified National Institute of Standards and Technology) [44] dataset, respectively.

The MMNN for XOR operation has two input units, three hidden units, and one output unit [50] and is trained by the approximately linear update scheme. The activation function is the binary function. The truth table of XOR operation is shown in Table 2. Variations of input and output voltages with training cycles of the XOR operation is shown in Fig. 7. After about 36 training cycles, the network correctly performs the XOR operation. The power consumption of crossbars for XOR operation is measured 2.18 mW in SPICE in the inference phase. But the total consumed energy is very low because the inference time is very short, which is nanosecond scale. If smaller input voltages and memristors

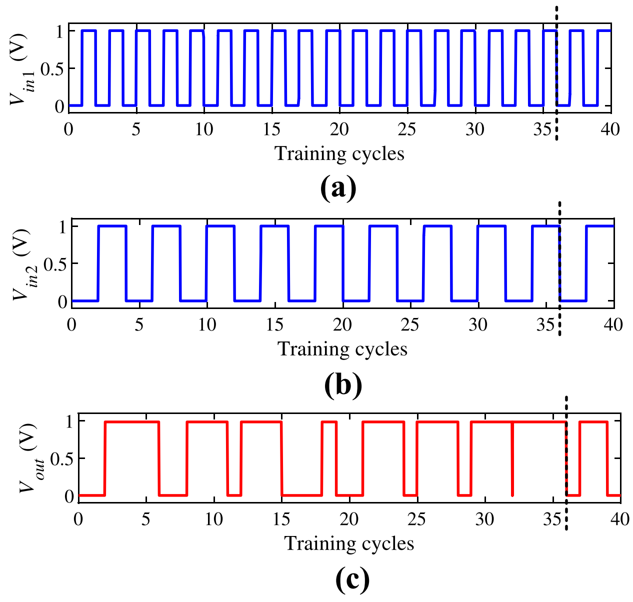

Fig. 7 Variations of input and output voltages with training cycles of the XOR operation. After about 36 training cycles, the network correctly performs the XOR operation (on the right side of the dotted line). **a** and **b** Input voltages. **c** Output voltages

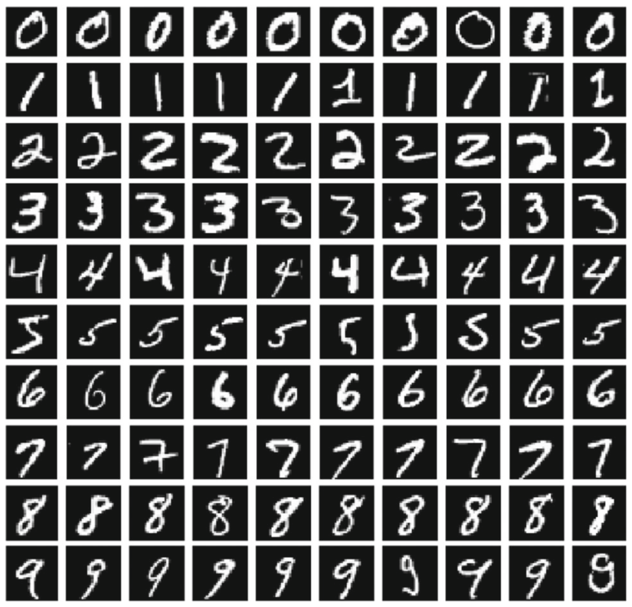

Fig. 8 Samples in each class of MNIST dataset. 10 samples per row belong to one class

with larger resistance are adopted, the power consumption can be reduced further.

The MMNN for digits recognition on MNIST has 784 input units, 256 hidden units, and 10 output units. The MNIST dataset contains handwritten digits from 0 to 9. There are total 60,000 training samples and 10,000 test samples of ten classes. Samples of each class are shown in Fig. 8. The input values are first converted to voltages among

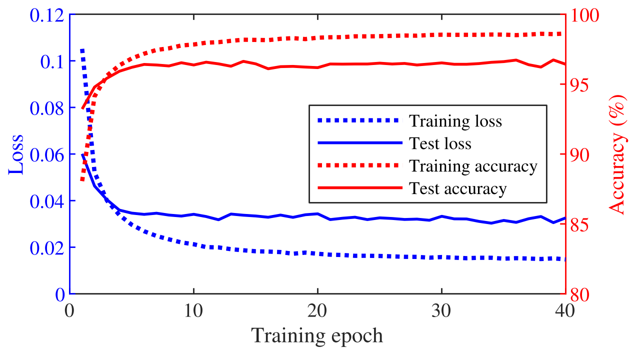

Fig. 9 Training accuracy, training loss, test accuracy, and test loss of MMNN trained by the fixed-voltage update scheme

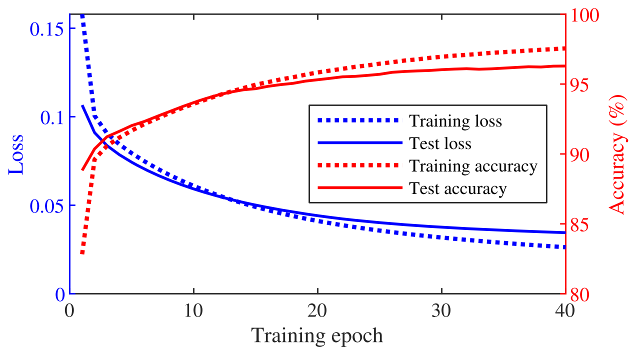

Fig. 10 Training accuracy, training loss, test accuracy, and test loss of MMNN trained by the approximately linear update scheme

Table 3 The architecture of MCNN

Layer	Size
Conv1	5×5, 6, $s = 1$
Avgpool1	2×2, $s = 2$
Conv2	5×5, 12, $s = 1$
Avgpool2	2×2, $s = 2$
FC	10

$[-V_r, V_r]$ through digital to analog converters (DACs), and then they are input to the memristor crossbars. The curves of training accuracy, training loss, test accuracy, and test loss versus training epochs under the two kinds of weight update schemes are shown in Figs. 9, 10, respectively. The classification accuracy of the fixed-voltage update is 96.42% and that of the approximately linear update is 96.29%.

Results of MCNN

The architecture of MCNN in simulations is shown in Table 3 [39]. Conv1 is the first convolutional layer and Avgpool1 is the first average pooling layer. $5 \times 5, 6, s = 1$ means that the kernel size is 5×5, the number of output channels is 6, and the convolution stride is 1.

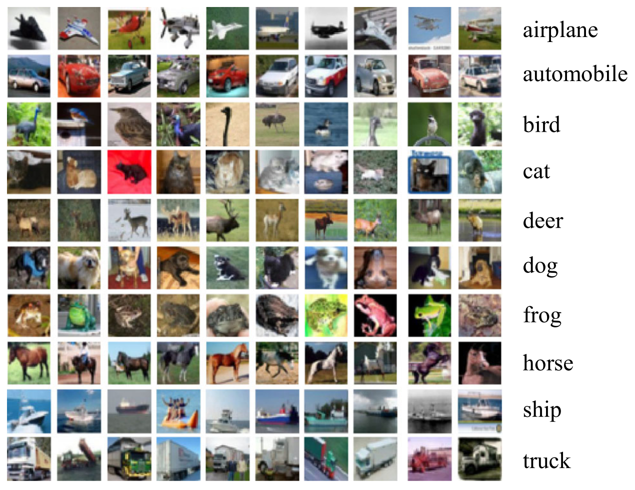

Fig. 11 Samples in each class of CIFAR-10 dataset. 10 samples per row belong to the same class

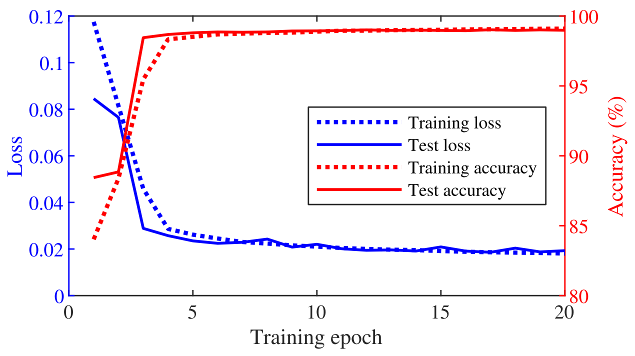

Fig. 12 Training loss, training accuracy, test loss, and test accuracy of MCNN on MNIST

MNIST and CIFAR-10 [16] datasets are adopted to substantiate the effectiveness of MCNN. CIFAR-10 is a widely used benchmark for image recognition. It contains 50,000 color training images and 10,000 test images of 10 classes, and samples of each class are shown in Fig. 11. The classification results of MCNN trained by the approximately linear update on the two datasets are shown in Figs. 12, 13, respectively. The final test accuracies of MNIST and CIFAR-10 are about 98.98% and 60.38%, respectively. MCNN is also trained by the fixed-voltage update scheme on MNIST and the test accuracy is 97.82%.

Results analysis

Classification results of MMNN and MCNN on MNIST are listed in Tables 4 and 5 and they are obtained by running the multiple cross-validation. It is seen that MCNN has better results than MMNN, and it can also be seen that the approximately linear update performs better than the fixed-voltage update. The confusion matrices of classi-

Table 4 Classification Results of MMNN

Dataset	Update scheme	Macro/ micro precision	Macro/ micro F1	Macro/ micro recall	Kappa coefficient	Accuracy
MNIST	Fixed-voltage	95.25/95.25	95.21/95.25	95.22/95.25	94.72	96.25
	Approximately linear	96.52/96.54	96.51/96.54	96.51/96.54	96.15	96.54
Fashion-MNIST	Fixed-voltage	84.34/84.07	84.07/84.07	83.97/84.07	82.30	84.07
	Approximately linear	86.74/86.88	86.88/86.88	86.76/86.88	85.42	86.88

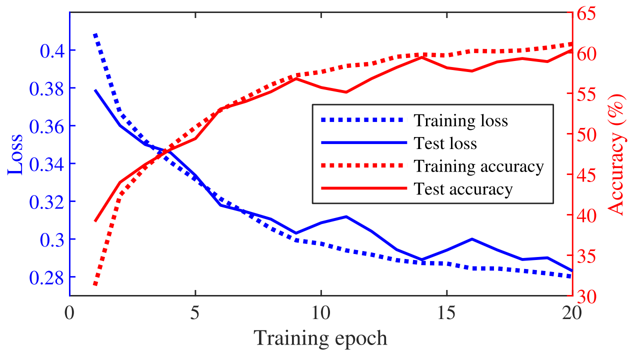

Fig. 13 Training loss, training accuracy, test loss, and test accuracy of MCNN on CIFAR-10

fication results of MNIST and Fashion-MNIST obtained by MMNN and MCNN are shown in Fig. 14.

Robustness analysis

For the fixed-voltage update, the duration of the writing voltage has an impact on the performance. Test errors of MMNN trained on MNIST by writing voltages with different pulse durations are shown in Fig. 15. In Fig. 15, the duration time is that of the voltage to decrease conductance. It is seen that the test error becomes large if the pulse duration is very large.

Because of the inherent characteristics of memristors, there are cycle-to-cycle (C2C) and device-to-device (D2D) variations in conductance adjustment. And the errors of writing voltages also result in conductance variations. To evaluate impacts of these variations on the performance of the networks, Gaussian noises with means 0 and standard deviations from 0 to 12% of the conductance value are considered as conductance variations in the approximately linear update. The conductance value after updating is $G_{new} = (G_{old} + \Delta G)(1 + s)$ [15], where G_{old} is the conductance before update and s is the noise level. Test errors of MMNN and MCNN under different variation levels are shown in Figs. 16, 17 respectively. It is seen that as the variation degree increases, the test error increases.

Computing complexity analysis

In the FC layer, the computation complexity of the vector-matrix multiplication is generally $O(N^2)$. In the memristor crossbar-based FC layer, the vector-matrix multiplication is performed with complexity $O(1)$. Activation functions are also performed at the same time in the circuit. In the convolutional layer, an efficient way to perform the convolutional operation is to convert it to matrix multiplication whose complexity is generally $O(N^3)$. The conversion is also needed for

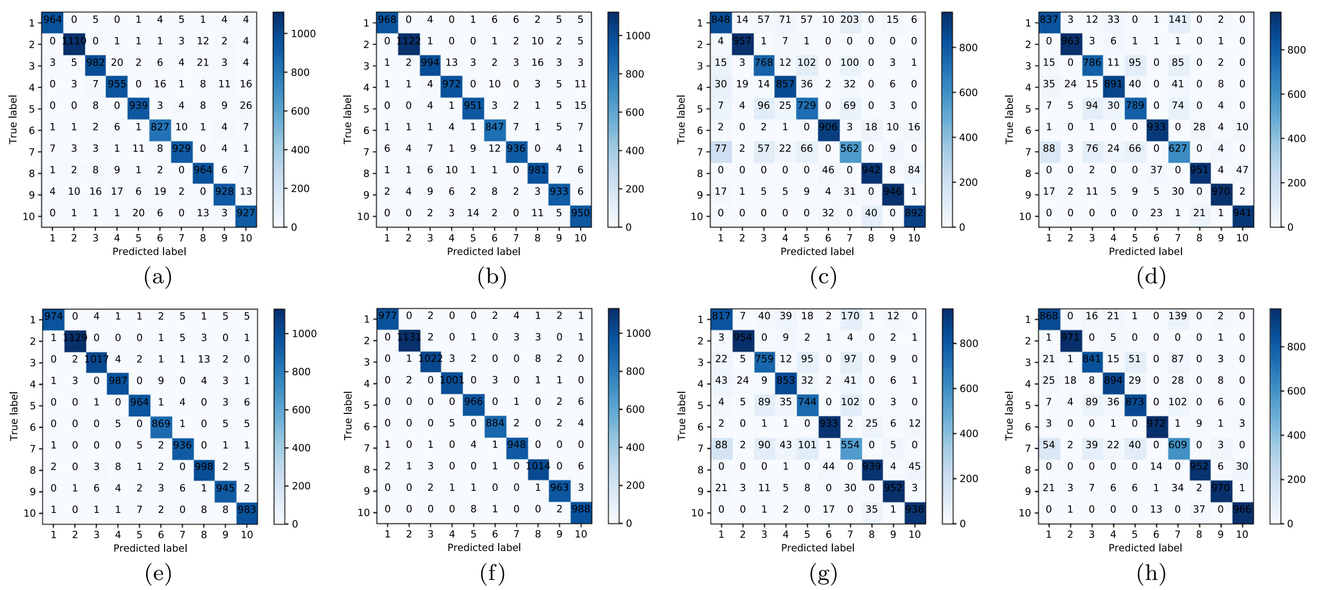

Fig. 14 Confusion matrices of classification results. **a** The confusion matrix of the classification result of MNIST obtained by MMNN trained by the fixed-voltage update scheme (expressed as MNIST-MMNN-fixed-voltage update). **b** MNIST-MMNN-approximately linear update. **c** Fashion-MNIST-MMNN-fixed-voltage update. **d** Fashion-MNIST-MMNN-approximately linear update. **e** MNIST-MCNN-fixed-voltage update. **f** MNIST-MCNN-approximately linear update. **g** Fashion-MNIST-MCNN-fixed-voltage update. **h** Fashion-MNIST-MCNN-approximately linear update

Table 5 Classification results of MCNN

Dataset	Update scheme	Macro/ micro precision	Macro/ micro F1	Macro/ micro recall	Kappa coefficient	Accuracy
MNIST	Fixed-voltage	98.02/98.02	97.99/98.02	98.00/98.02	97.80	98.02
	Approximately linear	98.94/98.94	98.93/98.94	98.93/98.94	98.82	98.94
Fashion-MNIST	Fixed-voltage	84.28/84.43	84.43/84.43	84.31/84.43	82.70	84.43
	Approximately linear	89.11/89.16	89.16/89.16	88.97/89.16	87.96	89.16
CIFAR-10	Fixed-voltage	49.62/50.12	50.12/50.12	49.69/50.12	44.58	50.12
	Approximately linear	61.45/60.63	60.63/60.63	59.81/60.63	56.26	60.63

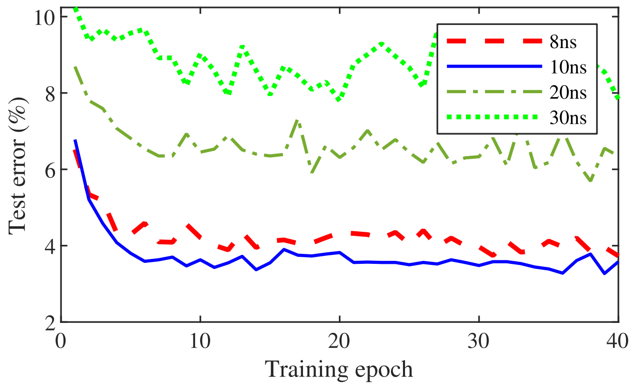

Fig. 15 Test errors of MMNN trained on MNIST by writing voltages with different pulse durations in the fixed-voltage update scheme

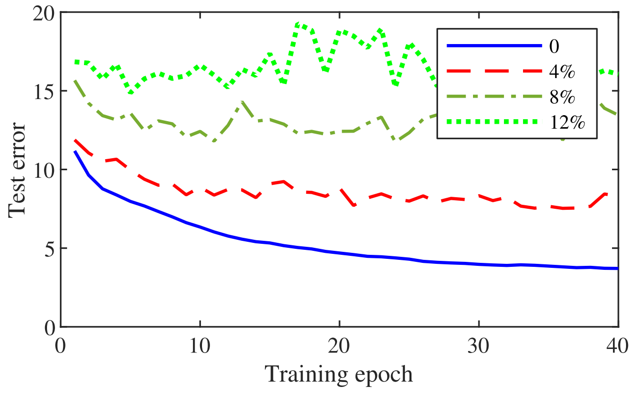

Fig. 16 Test errors of MMNN trained through the approximately linear update scheme on MNIST under different conductance variations

the memristor crossbar-based convoltional operation, and it is realized outside the crossbar. The advantage of the memristor crossbar-based convolutional operation is that it reduces the complexity of the matrix multiplication to $O(N)$. In the average pooling layer, the average pooling operation is performed with complexity $O(1)$. The weights are stored in the conductance of memristors and the vector-matrix multiplication is performed in-memory. The intermediate data of the convolutional operation is stored in the external storage which could also be realized by the memristor array.

Comparisons

Comparisons of MMNN with software-based MNN and other memristor-based MNNs [19,41,48,50] are shown in Table 6. Comparisons of MCNN with software-based CNN and other memristor-based CNNs [39,41,49] are shown in Table 7. It shows that the MMNN and MCNN built by the presented memristor-based crossbars and trained in-situ by the two kinds of weight update schemes have advantages in circuit functions and classification results compared with other memristor-based neural network circuits.

Table 6 Comparisons of MMNN with software-based and other memristor-based MNNs

	Software-based MNN	Memristor-based MNN in [19]*	Memristor-based MNN in [41]	Memristor-based MNN in [48]	MQ-MNN [49]	Memristor-based MNN in [50]	MMNN
Synapse structure	–**	2×1T1M	2×1T1M	2M	1M	1M	1M
Activation function	Sigmoid	ReLU	Pseudo sigmoid	Binary function	Binary function	Binary function	Pseudo sigmoid
Training mode	–	In-situ	Ex-situ	–	In-situ	In-situ	In-situ
Training method	BP and fixed update value/ original update value	BP	BP	SBP	BP	BP	BP and fixed-voltage update/ approximately linear update
Loss function	MSE	Softmax and cross-entropy loss	MSE	MSE	MSE	MSE	MSE
Test accuracy on MNIST	96.98%/ 97.32%	97.3±0.4%	–	94.5%	–	–	96.25%/ 96.54%

* The memristor crossbar is physically implemented and the activation functions are implemented in software
** "–" means that the indicator is not applicable or the related information is not provided in that paper

Table 7 Comparisons of MCNN with software-based and other memristor-based CNNs

	Software-based CNN	Memristor-based CNN in [39]	Memristor-based CNN in [41]	Memristor-based CNN in [46]*	MQ-CNN [49]	MCNN
Synapse structure	–	2M	2M	2×1T1M	1M	1M
Activation function	ReLU	Pseudo sigmoid	Pseudo sigmoid	ReLU	–	Pseudo ReLU
Training sample number	60,000	10,000	–	60,000	60,000	60,000
Test sample number	10,000	500	–	10,000	10,000	10,000
Epoch numbers	40	10	–	550	40	40
Training mode	–	Ex-situ	Ex-situ	Ex-situ and in-situ	–	In-situ
Training method	BP and fixed update value/original update value	BP	BP	BP	BP	BP and fixed-voltage update/ approximately linear update
Loss function	MSE	–	MSE	Softmax and cross-entropy loss	–	MSE
Test accuracy on MNIST	99.03%/99.12%	91.8%	≈92%	96.19%	98.97%	98.02%/98.94%

*The memristor crossbar is physically implemented and the activation functions are realized by running the codes on ARM cores

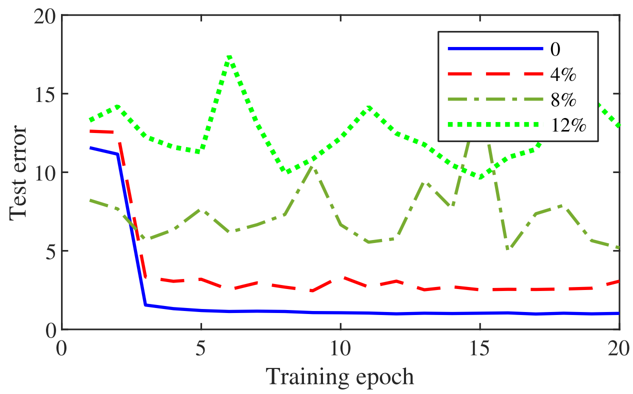

Fig. 17 Test errors of MCNN trained through the approximately linear update scheme on MNIST under different conductance variations

Conclusions

This paper presents memristor crossbar architectures for implementation of DNNs, which include architectures for the FC layer, convolutional operation, and average pooling operation. MMNN and MCNN are built to evaluate the performance of these memristor crossbar architectures. The networks are in-situ trained by two kinds of weight update schemes, which are the fixed-voltage update and the approximately linear update, and simulation results show that the networks trained by the weight update schemes result in satisfying performance. The robustness of MMNN and MCNN to conductance variations of memristors is also analyzed. In summary, the memristor-based DNNs constructed by presented memristor crossbars perform satisfactorily in pattern recognition tasks and have certain robustness to imperfections of hardware.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Adam K, Smagulova K, James AP (2018) Memristive LSTM network hardware architecture for time-series predictive modeling

problems. In: IEEE Asia Pacific conference on circuits and systems. Chengdu, China, pp 459–462
2. Adam K, Smagulova K, Krestinskaya O, James AP (2018) Wafer quality inspection using memristive LSTM, ANN, DNN and HTM. In: IEEE electrical design of advanced packaging and systems symposium. Chandigarh, India
3. Adhikari SP, Yang C, Kim H, Chua LO (2012) Memristor bridge synapse-based neural network and its learning. IEEE Trans Neural Netw Learn Syst 23(9):1426–1435
4. Alibart F, Zamanidoost E, Strukov DB (2013) Pattern classification by memristive crossbar circuits using ex situ and in situ training. Nat Commun 4:4
5. Cantley KD, Subramaniam A, Stiegler HJ, Chapman RA, Vogel EM (2012) Neural learning circuits utilizing nano-crystalline silicon transistors and memristors. IEEE Trans Neural Netw Learn Syst 23(4):565–573
6. Chua LO (1971) Memristor-the missing circuit element. IEEE Trans Circ Theory 18(5):507–519
7. Di Marco M, Forti M, Pancioni L (2017) Memristor standard cellular neural networks computing in the flux-charge domain. Neural Netw 93:152–164
8. Gokmen T, Onen M, Haensch W (2017) Training deep convolutional neural networks with resistive cross-point devices. Front Neurosci 11:538
9. Gokmen T, Rasch M, Haensch W (2018) Training LSTM networks with resistive cross-point devices. URL http://arxiv.org/abs/1806.00166
10. Hasan R, Taha TM (2014) Enabling back propagation training of memristor crossbar neuromorphic processors. In: International joint conference on neural networks. Beijing, China, pp 21–28
11. Hasan R, Taha TM, Yakopcic C (2017) On-chip training of memristor crossbar based multi-layer neural networks. Microelectron J 66:31–40
12. Hu M, Graves CE, Li C, Li Y, Ge N, Montgomery E et al (2018) Memristor-based analog computation and neural network classification with a dot product engine. Adv Mater 30(9):1
13. Krestinskaya O, Ibrayev T, James AP (2018) Hierarchical temporal memory features with memristor logic circuits for pattern recognition. IEEE Trans Comput Aided Des Integr Circ Syst 37(6):1143–1156
14. Krestinskaya O, James AP (2018) Feature extraction without learning in an analog spatial pooler memristive-cmos circuit design of hierarchical temporal memory. Analog Integr Circ Signal Process 95(3):457–465
15. Krestinskaya O, Salama KN, James AP (2019) Learning in memristive neural network architectures using analog backpropagation circuits. IEEE Trans Circ Syst I Reg Papers 66(2):719–732
16. Krizhevsky A (2009) Learning multiple layers of features from tiny images
17. Kvatinsky S, Friedman EG, Kolodny A, Weiser UC (2013) TEAM: threshold adaptive memristor model. IEEE Trans Circ Syst I Reg Papers 60(1):211–221
18. Kvatinsky S, Ramadan M, Friedman EG, Kolodny A (2015) VTEAM: a general model for voltage-controlled memristors. IEEE Trans Circ Syst II Exp Briefs 62(8):786–790
19. Li C, Belkin D, Li Y, Yan P, Hu M, Ge N et al (2018) Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. Nat Commun 9(1):1
20. Li C, Wang Z, Rao M, Belkin D, Song W, Jiang H et al (2019) Long short-term memory networks in memristor crossbar arrays. Nat Mach Intell 1(1):49
21. Liu J, Li Z, Tang Y, Hu W, Wu J (2020) 3D convolutional neural network based on memristor for video recognition. Pattern Recogn Lett 130:116–124
22. Liu X, Huang Y, Zeng Z, Wunsch DC II (2020) Memristor-based HTM spatial pooler with on-device learning for pattern recognition. IEEE Trans Syst Man Cybern Syst. https://doi.org/10.1109/TSMC.2020.3035612
23. Liu X, Zeng Z II, DCW, (2020) Memristor-based LSTM network with in situ training and its applications. Neural Netw 131:300–311
24. Liu X, Zeng Z, Wen S (2016) Implementation of memristive neural network with full-function Pavlov associative memory. IEEE Trans Circ SystI Reg Papers 63(9):1454–1463
25. Pershin Y, Di Ventra M (2010) Experimental demonstration of associative memory with memristive neural networks. Neural Netw 23(7):881–886
26. Shafiee A, Nag A, Muralimanohar N, Balasubramonian R, Strachan JP, Hu M, Williams RS, Srikumar V (2016) ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: ACM/IEEE 43rd annual international symposium on computer architecture. Seoul, South Korea, pp 14–26
27. Silva F, Sanz M, Seixas J, Solano E, Omar Y (2020) Perceptrons from memristors. Neural Netw 122:273–278
28. Smagulova K, Adam K, Krestinskaya O, James AP (2018) Design of CMOS-memristor circuits for LSTM architecture. In: IEEE International conferences on electron devices and solid-state circuits. Shenzhen, China
29. Smagulova K, Krestinskaya O, James AP (2018) A memristor-based long short term memory circuit. Analog Integ Circ Signal Process 95(3):467–472
30. Soudry D, Di Castro D, Gal A, Kolodny A, Kvatinsky S (2015) Memristor-based multilayer neural networks with online gradient descent training. IEEE Trans Neural Netw Learn Syst 26(10):2408–2421
31. Strukov DB, Snider GS, Stewart DR, Williams RS (2008) The missing memristor found. Nature 453(7191):80
32. Truong SN, Min KS (2014) New memristor-based crossbar array architecture with 50-% area reduction and 48-% power saving for matrix-vector multiplication of analog neuromorphic computing. J Semicond Technol Sci 14(3):356–363
33. Truong SN, Van Pham K, Min KS (2018) Spatial-pooling memristor crossbar converting sensory information to sparse distributed representation of cortical neurons. IEEE Trans Nanotechnol 17(3):482–491
34. Wang Z, Wang X (2018) A novel memristor-based circuit implementation of full-function Pavlov associative memory accorded with biological feature. IEEE Trans Circ Syst I Reg Papers 65(7):2210–2220
35. Wen S, Wei H, Yang Y, Guo Z, Zeng Z, Huang T, Chen Y (2019) Memristive LSTM network for sentiment analysis. IEEE Trans Syst Man Cybern Syst. https://doi.org/10.1109/TSMC.2019.2906098
36. Wen S, Wei H, Zeng Z, Huang T (2018) Memristive fully convolutional network: An accurate hardware image-segmentor in deep learning. IEEE Trans Emerg Top Comput Intell 2(5):324–334
37. Wen S, Xie X, Yan Z, Huang T, Zeng Z (2018) General memristor with applications in multilayer neural networks. Neural Netw 103:142–149
38. Xie X, Wen S, Zeng Z, Huang T (2018) Memristor-based circuit implementation of pulse-coupled neural network with dynamical threshold generators. Neurocomputing 284:10–16
39. Yakopcic C, Alom MZ, Taha TM (2016) Memristor crossbar deep network implementation based on a convolutional neural network. In: International joint conference on neural networks. Vancouver, Canada, pp 963–970
40. Yakopcic C, Alom MZ, Taha TM (2017) Extremely parallel memristor crossbar architecture for convolutional neural network implementation. In: International joint conference on neural networks. Anchorage, USA, pp 1696–1703
41. Yakopcic C, Hasan R, Taha TM (2015) Memristor based neuromorphic circuit for ex-situ training of multi-layer neural network algorithms. In: International joint conference on neural networks. Killarney, Ireland

42. Yang L, Zeng Z, Shi X (2019) A memristor-based neural network circuit with synchronous weight adjustment. Neurocomputing 363:114–124
43. Yang L, Zeng Z, Wen S (2018) A full-function Pavlov associative memory implementation with memristance changing circuit. Neurocomputing 272:513–519
44. Yann L, Léon B, Yoshua B, Patrick H (1998) Gradient-based learning applied to document recognition. Proceed IEEE 86(11):2278–2324
45. Yao P, Wu H, Gao B, Eryilmaz SB, Huang X, Zhang W et al (2017) Face classification using electronic synapses. Nature Commun 8:1
46. Yao P, Wu H, Gao B, Tang J, Zhang Q, Zhang W, Yang JJ, Qian H (2020) Fully hardware-implemented memristor convolutional neural network. Nature 577(7792):641–646
47. Zamanidoost E, Bayat FM, Strukov D, Perceptron AMl (2015) Manhattan rule training for memristive crossbar circuit pattern classifiers. In: IEEE International symposium on intelligent signal processing, pp 1–6. Siena, Italy
48. Zhang Q, Wu H, Yao P, Zhang W, Gao B (2018) Sign backpropagation: an on-chip learning algorithm for analog RRAM neuromorphic computing systems. Neural Netw 108:217–223
49. Zhang Y, Cui M, Shen L, Zeng Z (2019) Memristive quantized neural networks: a novel approach to accelerate deep learning on-chip. IEEE Trans Cybern 1:1–13
50. Zhang Y, Wang X, Friedman EG (2018) Memristor-based circuit design for multilayer neural networks. IEEE Trans Circ Syst I Reg Papers 65(2):677–686
51. Zhang Y, Wang X, Li Y, Friedman EG (2017) Memristive model for synaptic circuits. IEEE Trans Circ Syst II Exp Briefs 64(7):767–771
52. Zhang Y, Zeng Z, Wen S (2014) Implementation of memristive neural networks with spike-rate-dependent plasticity synapses. In: International joint conference on neural networks, pp 2226–2233
53. Zhu S, Wang L, Duan S (2017) Memristive pulse coupled neural network with applications in medical image processing. Neurocomputing 227:149–157
54. Zyarah AM, Kudithipudi D (2019) Neuromemrisitive architecture of HTM with on-device learning and neurogenesis. ACM J Emerg Technol Comput Syst 15(3):24

Xiaoyang Liu received the B.S. degree from the College of Electrical and Information Engineering, Hunan University, Changsha, China, in 2014. He is currently working toward the Ph.D. degree in the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, and also with the Key Laboratory of Image Processing and Intelligent Control of Education Ministry of China, Wuhan, China. His current research interests include memristor-based artificial neural networks and image recognition.

Zhigang Zeng (F'20) received the Ph.D. degree in systems analysis and integration from Huazhong University of Science and Technology, Wuhan, China, in 2003. He is currently a Professor with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China, and also with the Key Laboratory of Image Processing and Intelligent Control of the Education Ministry of China, Wuhan, China. He has published more than 100 international journal papers. His current research interests include theory of functional differential equations and differential equations with discontinuous right-hand sides, and their applications to dynamics of neural networks, memristive systems, and control systems. Dr. Zeng has been an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS from 2010 to 2011, the IEEE TRANSACTIONS ON CYBERNETICS since 2014, the IEEE TRANSACTIONS ON FUZZY SYSTEMS since 2016. He has been a member of the Editorial Board of Neural Networks since 2012, Cognitive Computation since 2010, Applied Soft Computing since 2013.