

A fast training method for memristor crossbar based multi-layer neural networks

Raqibul Hasan¹  · Tarek M. Taha¹ · Chris Yakopcic¹

Received: 26 January 2017 / Accepted: 25 September 2017 / Published online: 5 October 2017
© Springer Science+Business Media, LLC 2017

Abstract Memristor crossbar arrays carry out multiply–add operations in parallel in the analog domain which is the dominant operation in a neural network application. On-chip training of memristor neural network systems have the significant advantage of being able to get around device variability and faults. This paper presents a novel technique for on-chip training of multi-layer neural networks implemented using a single crossbar per layer and two memristors per synapse. Using two memristors per synapse provides double the synaptic weight precision when compared to a design that uses only one memristor per synapse. Proposed system utilizes a novel variant of the back-propagation (BP) algorithm to reduce both circuit area and training time. During training, all the memristors in a crossbar are updated in four steps in parallel. We evaluated the training of the proposed system with some nonlinearly separable datasets through detailed SPICE simulations which take crossbar wire resistance and sneak-paths into consideration. The proposed training algorithm trained the nonlinearly separable functions with a slight loss in accuracy compared to training with the traditional BP algorithm.

Keywords Neural networks · Memristor crossbars · Training algorithm · On-chip training

✉ Raqibul Hasan
hasanm1@udayton.edu

Tarek M. Taha
tarek.taha@udayton.edu

Chris Yakopcic
cyakopcic1@udayton.edu

¹ Department of Electrical and Computer Engineering,
University of Dayton, Dayton, OH, USA

1 Introduction

Reliability and power consumption are among the main obstacles for continued performance improvement in future computing systems. Embedded neural network based processing systems have significant advantages to offer, such as the ability to solve complex problems while consuming very little power and area [1, 2]. Memristors [3, 4] have received significant attention as a potential building block for neuromorphic systems [5, 6]. In these systems memristors are used in a crossbar structure. Memristor devices in a crossbar structure can evaluate many multiply–add operations in parallel in the analog domain very efficiently (these are the dominant operations in neural networks). This enables highly dense neuromorphic system with great computational efficiency [1].

It is necessary to have an efficient training system for memristor neural network based systems. Two approaches for training are off-chip training and on-chip training. The key benefit of off-chip training is that any training algorithm can be implemented in software and run on powerful computer clusters. Memristor crossbars are difficult to model in software due to sneak paths and device variations [7, 8]. On-chip training has the advantage that it can take into account variations between devices and can use the full analog range of the device (as opposed to a set of discrete resistances that off-chip training typically targets).

This paper presents circuits for on-chip training of memristor crossbars that utilize two memristors per synapse. The use of two memristors per synapse has significant advantages over using a single memristor per synapse. Most recent memristor crossbar circuit fabrications for neuromorphic computing have been using two memristors per synapse [9, 10]. Using two memristors per synapse provides double the synaptic weight precision

when compared to a design that uses only one memristor per synapse. This can enable better training of the neural networks [13].

Existing work in on-chip training circuits for memristor crossbars include [11, 27]. Soudry et al. [11] examined on-chip gradient descent based training of memristor crossbars with a single memristor per synapse. They did not consider the training of systems with two memristors per synapse and updated synaptic weights serially column by column. Work in [27] proposed to apply a variable amplitude training pulse and update a crossbar in 4 steps in parallel. Their approach needs to evaluate $\log[e]$ for multiple neurons in parallel (their “e” is a function of the derivative of the activation function). They did not detail design of the required circuits to evaluate these operations.

This paper presents on-chip training circuits for multi-layer neural networks implemented using a single crossbar per layer and two memristors per synapse. We utilize a novel variant of the back-propagation (BP) algorithm [14] to reduce both circuit area and training time. The proposed system requires four steps to update all the memristors in a crossbar in parallel (for a training instance/data). This enables faster training of neural networks. We design novel circuit to generate training pulse of variable amplitude and duration.

The rest of the paper is organized as follows: Sect. 2 describes related work in the area. Section 3 demonstrates memristor based neuron circuit and neural network design. Section 4 describes the proposed training algorithm and its hardware implementation. Sections 5 and 6 describe the experimental setup and results respectively. Finally, Sect. 7 concludes the paper.

2 Related work

Zamarreño-Ramos et al. [20] examined how a memristor grid can implement a highly dense spiking neural network and used it for visual image processing. They examined STDP training to implement spiking neural networks. Alibart et al. [9], Chabi et al. [21], and Starzyk et al. [22] demonstrated pattern classification using a single layer perceptron network implemented utilizing a memristive crossbar circuit. Nonlinearly separable problems were not studied in these works.

Memristor bridge circuits have been proposed [23, 24] where small groups of memristors (either 4 or 5) are used to store a synaptic weight. One of the advantages of these bridge circuits is that either a positive or negative weight can be stored based on the sensed voltage. Adhikari et al. [24] examined multi-layer neural network training using memristor bridge circuits. They utilized random weight update rule which does not require error back propagation.

Their results showed that the training using random weight update rule converges slowly than the training using the BP algorithm.

Soudry et al. [11] proposed gradient decent based learning on a memristor crossbar neural network. They utilized two transistors and one memristor per synapse. Synaptic weight precision of the proposed implementations are two times more than this design. The hardware implementation of their approach would require ADCs and set of multi bit buffers. An error back propagation step would require multipliers and a set of DACs. For a layer with m inputs and n neurons, a weight update requires $O(m + n)$ operations in [11] while our proposed training approach requires $O(c)$ operations (c is a constant).

Work in [12] proposed using two crossbars for the same weight values. Since the switching characteristic of a memristor often contains some degree of noise, it would be difficult to store an exact copy of a memristor crossbar without a complex feedback write mechanism. Our proposed work is able to apply a variable pulse width during the weight update which is not possible in the system in Li et al. [12].

Training a multi-layer neural network requires the output layer error to be back propagated to the hidden layer neurons. Paper [25, 26] examined training of multi-layer neural networks using a training algorithm named “Manhattan Rule”. They did not detail the error back propagation step and design of the training pulse generation circuitry. Works in [9, 25, 26] update the weights by a constant amount, applying a constant amplitude and duration pulse. Work in [27] proposed to apply a variable amplitude pulse during training, but did not detail the required training circuit design. Their design is based on the assumption that conductance change, $\Delta G \propto \exp[V]$, which is probably a very rigid assumption (where V is the applied voltage across the memristor). Such assumption is not made in the proposed work. Kataeva et al. [27] proposed to update a crossbar in 4 steps in parallel, but did not detail circuit to evaluate $\log[e]$ for multiple neurons in parallel (their “e” is a function of the derivative of the activation function). How these analog values are stored and applied in parallel to a crossbar is not detailed in [27]. Our proposed design does not have these limitations.

3 Memristor crossbar based neuron circuit and neural network implementation

3.1 Neuron circuit

A neuron in a neural network performs two types of operations, (1) a dot product of the inputs x_1, \dots, x_n and the weights w_1, \dots, w_n , and (2) the evaluation of an activation

function. The dot product operation can be seen in Eq. (1). The activation function of a neuron is shown in Eq. (2). In a multi-layer feed forward neural network, a nonlinear differentiable activation function is desired (e.g. $\tan^{-1}(x)$).

$$DP_j = \sum_{i=1}^n x_i w_{ij} \tag{1}$$

$$y_j = f(DP_j) \tag{2}$$

Figure 1(a) shows the memristor based neuron circuit design utilized in this paper. This circuit has three inputs and one bias input (β). For each input, both the data and its complemented form are applied to the neuron circuit. Here, a complemented input (such as \bar{A}) has the same magnitude but opposite polarity as the original input (A). For the input pair A and \bar{A} in Fig. 1(a), if $\sigma_A^+ > \sigma_{\bar{A}}$, then the synapse corresponding to input A has a positive weight. Likewise if $\sigma_A^+ < \sigma_{\bar{A}}$, then this synapse has a negative weight. This applies to each of the inputs. The output of the inverter pair at the bottom of the circuit represents the neuron output. Assume that the conductance of the memristors of Fig. 1(a) from top to bottom are $\sigma_{A+}, \sigma_{A-}, \dots, \sigma_{\beta+}, \sigma_{\beta-}$. In an ideal case the potential at the first inverter input (DP_j) can be expressed by Eq. (3) which indicates that this circuit is essentially carrying out a set of multiply–add operations in parallel in the analog domain. In Eq. (3) the denominator is always greater than zero and works as a scaling factor.

$$DP_j = \frac{A(\sigma_{A+} - \sigma_{A-}) + \dots + \beta(\sigma_{\beta+} - \sigma_{\beta-})}{\sigma_{A+} + \sigma_{A-} + \dots + \sigma_{\beta+} + \sigma_{\beta-}} \tag{3}$$

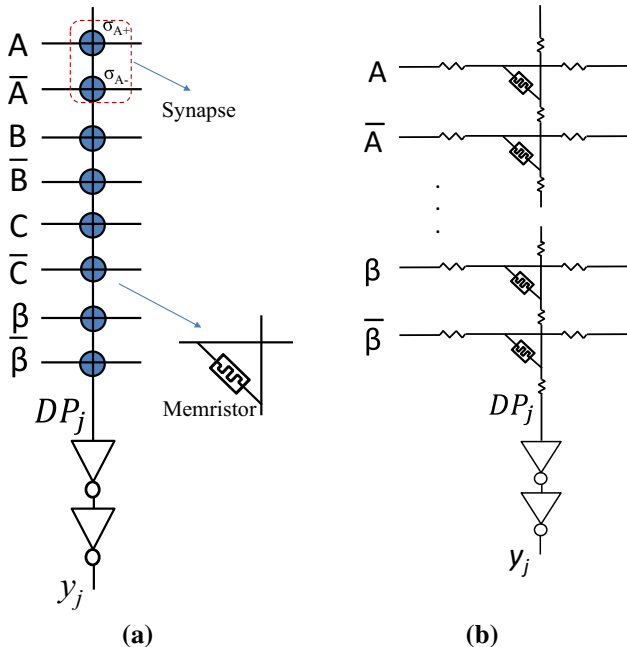


Fig. 1 Memristor based neuron circuit. A, B, C are the inputs and y_j is the output

Neuron circuit in Fig. 1(a) is implementing the activation function using a pair of CMOS inverters. The power rails to the inverters are $V_{DD} = 0.5 \text{ V}$ and $V_{SS} = -0.5 \text{ V}$, hence it provides a neuron output thresholded at 0 V. Our experimental evaluations consider memristor crossbar wire resistance. The schematics of the memristor based neuron circuits considering wire resistance are shown in Fig. 1(b).

3.2 Synaptic weight precision

The precision of memristor based synaptic weights depends on the number of memristors used for each synapse and the resistance (or conductance) range of the memristor device. Assume that the maximum conductance of the memristor device is σ_{max} and the minimum conductance is σ_{min} . In a design using only a single memristor per synapse, $\bar{\sigma}$ defines the separation between positive and negative weights [11]. Table 1 shows that the range of synaptic weights when using two memristors per synapse is two times that of a single memristor per synapse design.

3.3 Multi-layer circuit design

The implementation of a nonlinear classifier requires a multi-layer neural network. Figure 2(a) shows a simple two layer feed forward neural network with three inputs, two outputs, and six hidden layer neurons. Figure 2(b) shows a memristor crossbar based implementation of the neural network in Fig. 2(a), utilizing the neuron circuit shown in Fig. 1(a). There are two memristor crossbars in this circuit, each representing a layer of neurons. When the inputs are applied to a crossbar, the entire crossbar is processed in parallel within one cycle.

4 Memristor crossbar based multi-layer neural network for the proposed training algorithm

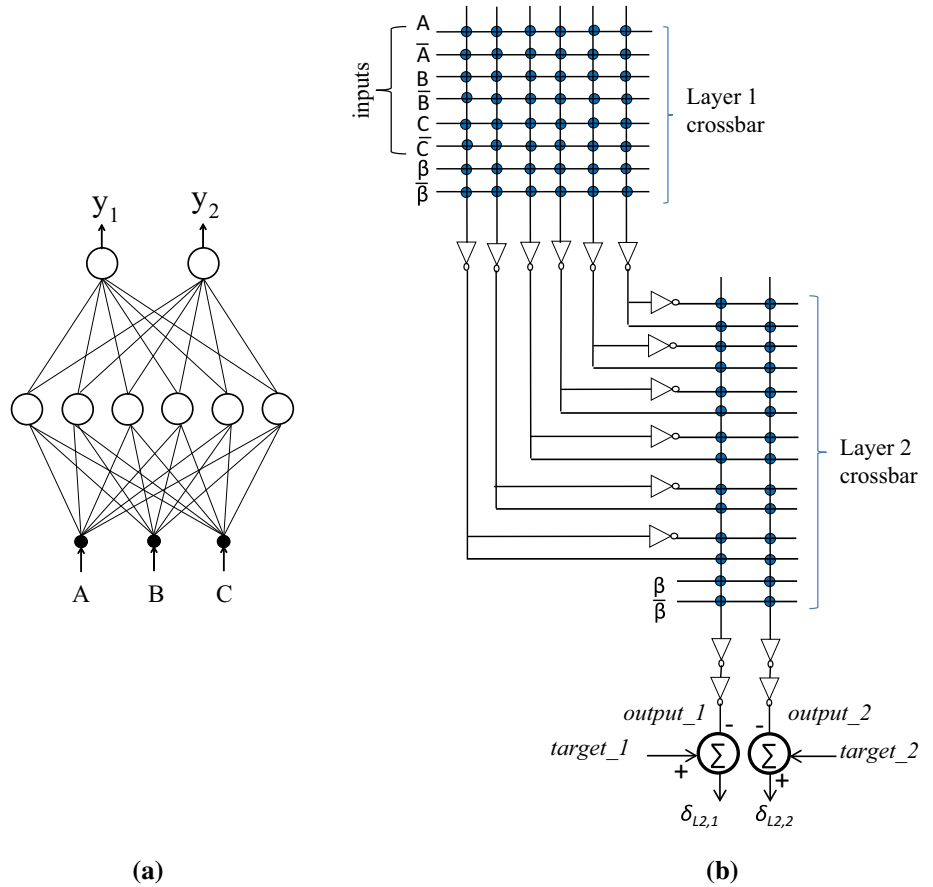
4.1 Proposed training algorithm

The hardware implementation of the exact BP algorithm is expensive as it requires ADC, DACs, lookup table (to evaluate derivative of the activation function), multiplier [11]. In this section a variant of the stochastic BP algorithm is proposed as a low cost hardware implementation of the algorithm. Proposed training algorithm enables weight update, for a layer of neurons, in four steps as opposed to two steps per crossbar column [11]. When implementing back propagation training in software, a commonly used activation function is one that is based on the inverse tangent as seen in Eq. (4) and its derivative in Eq. (5).

Table 1 Synaptic weight precision

	Two memristors per synapse	One memristor per synapse
Maximum weight	$\sigma_{max} - \sigma_{min}$	$\sigma_{max} - \bar{\sigma}$
Minimum weight	$\sigma_{min} - \sigma_{max}$	$\sigma_{min} - \bar{\sigma}$
Range	$2(\sigma_{max} - \sigma_{min})$	$\sigma_{max} - \sigma_{min}$

Fig. 2 a Two layer network for learning three input functions. **b** Schematic of the neural network shown in (a) for forward pass utilizing neuron circuit in Fig. 1(a)



$$f_2(x) = \frac{1}{\pi} \tan^{-1}(x) \tag{4}$$

$$\frac{df_2(x)}{dx} = \frac{1}{\pi} \times \frac{1}{1+x^2} = \frac{1}{\pi} g(x) \tag{5}$$

[where $g(x) = 1/(1 + x^2)$].

The proposed hardware implementation approximates these functions using analog hardware. The neuron activation function is implemented using the double inverter circuit in Fig. 1(a), and can be modeled using a threshold function as in Eq. (6). This allows for an implementation with higher speed, lower area, and lower energy.

$$f_{th}(x) = \begin{cases} 0.5, & x \geq 0 \\ -0.5, & x < 0 \end{cases} \tag{6}$$

To simplify the weight update equation for a low cost hardware implementation, $g(x)$ in Eq. (5) was approximated as the piecewise linear function $g_{th}(x)$ [see Eq. (7)].

$$g_{th}(x) = \begin{cases} 1 - |x|/2, & |x| < 1.9 \\ 0.05, & \text{else} \end{cases} \tag{7}$$

The proposed training algorithm for the altered back propagation algorithm is stated below:

1. Initialize the memristors with high random resistances.
2. For each input pattern x :
 - i. Apply the input pattern x to the crossbar circuit and evaluate DP_j values and outputs (y_j) of all neurons (hidden neurons and output neurons).
 - ii. For each output layer neuron j , calculate the error, δ_j , between the neuron output (y_j) and the target output (t_j).

$$\delta_j = \text{sgn}(t_j - y_j) \tag{8}$$

- iii. If $\sum_{j \in (\text{output neuron})} |\delta_j| = 0$ then goto step 2 for the next input pattern.
- iv. Back propagate the error for each hidden layer neuron j .

$$\delta_j = \text{sgn} \left(\sum_k \delta_k w_{kj} \right) \tag{9}$$

where neuron k is connected to the previous layer neuron j .

- v. Determine the amount, Δw , that each neuron’s synapses should be changed ($2\eta\pi$ is the learning rate):

$$\Delta w_j = 2\eta \times \delta_j \times g_{th}(DP_j) \times x \tag{10}$$

- 3. If the error in the output layer has not converged to a sufficiently small value, goto step 2.

In this algorithm, Eqs. (8)–(10) are different from the traditional back-propagation algorithm. Instead of evaluating the actual error, we evaluate a sign function of the error. It is much simpler to store a 2 bit value than an analog value, and this allows for a simpler hardware implementation. Additionally, we do not need expensive ADCs, DACs, lookup table, multiplier in this approach. Furthermore, for a layer of neurons, the weight update could be done in four steps which will enable faster training (detailed in Sect. 4.4).

4.2 Training algorithm comparison

To examine the functionality of the proposed training algorithm we have trained neural networks for different nonlinearly separable datasets: (a) 2 input XOR function, (b) 3 input odd parity function, (c) 4 input odd parity function, (d) Wine [16], (e) Iris [17] and, (f) MNIST [18]. We have trained the neural networks for these datasets in MATLAB utilizing the configurations shown in Table 2. A neural network configuration is described as $x \rightarrow y \rightarrow z$, where the network has x inputs, y hidden neurons, and z output neurons. We also trained these datasets using the traditional BP algorithm for comparison. For each dataset

both training approaches utilized the same network configurations, learning rates and the same maximum epoch counts. Figure 3 shows the mean squared error (MSE) obtained for different datasets using the proposed algorithm as well as the traditional back-propagation algorithm. We observe that the proposed algorithm converges to the minimum specified error.

For the 2 input XOR, 3 input odd parity, and 4 input odd parity functions, 100% recognition accuracy was achieved with both the proposed algorithm and the traditional BP approach. The proposed training algorithm can train nonlinearly separable functions with a slight loss in accuracy compared to when training with the BP algorithm. For the Wine, Iris, and MNIST datasets, recognition errors on test dataset using the proposed algorithm were 3.3, 5, and 4.9% respectively. Corresponding recognition errors using the BP algorithm were 3.3, 2, and 3.5% respectively. Our test errors for the MNIST dataset are slightly bigger than the published best recognition error, because instead of using softmax function in the output layer we are using binary threshold function (for ease of hardware implementation). The training curves for the proposed algorithm (in Fig. 3(d), (e) and (f)) are not as smooth as the curves obtained when using the BP algorithm because in the former one, we are utilizing a sign function of the errors.

4.3 Circuit implementation of the proposed training algorithm

Without loss of generality we will describe the circuit implementation of the back-propagation training algorithm for the neural network shown in Fig. 2(a). The implementation of the training circuit can be broken down into the following major steps:

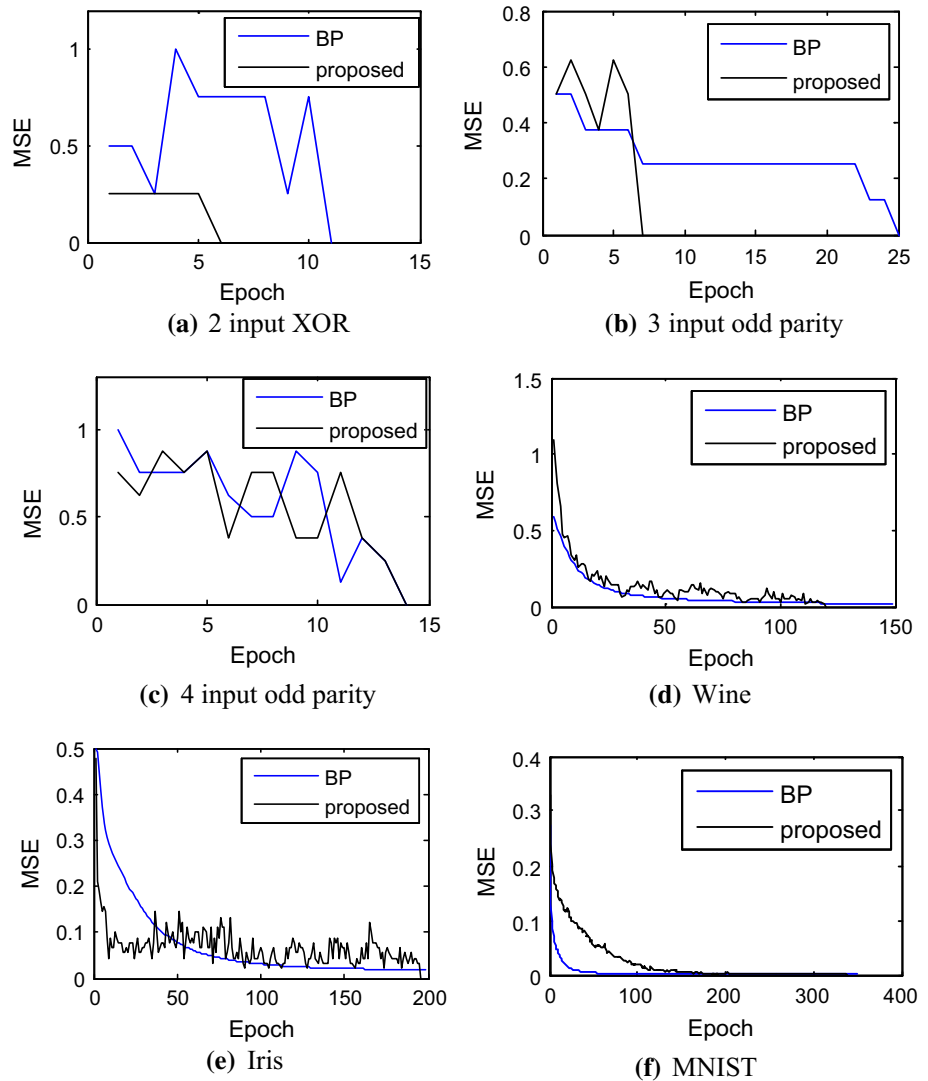
1. Apply inputs to layer 1 and record the layer 2 neuron outputs and errors.
2. Back-propagate layer 2 errors through the second layer weights and record the layer 1 errors.
3. Update the synaptic weights.

The circuit implementations of these steps are detailed below:

Table 2 Neural network configurations

Dataset	Neural network configurations	Number of training data	Learning rate
2 input XOR	2 → 5 → 1	4	0.1
3 input odd parity	3 → 7 → 1	8	0.07
4 input odd parity	4 → 12 → 1	16	0.07
Wine	13 → 20 → 3	118	0.05
Iris	4 → 15 → 3	99	0.05
MNIST	784 → 800 → 10	50,000	0.07

Fig. 3 Mean squared errors versus training epoch for both the proposed training algorithm and a traditional BP algorithm



Step 1: A set of inputs is applied to the layer 1 neurons, and the layer 2 neuron outputs are measured. This process is shown in Fig. 2(b). The terms $\delta_{L2,1}$ and $\delta_{L2,2}$ are the error terms and are based on the difference between the observed outputs (t_i) and the expected outputs (y_i). The error values could be + 1, - 1 or 0. Thus these errors can easily be recorded in binary form for later use.

Step 2: The layer 2 errors ($\delta_{L2,1}$ and $\delta_{L2,2}$) are applied to the layer 2 weights as shown in Fig. 4 to generate the layer 1 errors ($\delta_{L1,1}$ to $\delta_{L1,6}$). Assume that the synaptic weights associated with input i , neuron j (second layer neuron) is $w_{ij} = \sigma_{ij}^+ - \sigma_{ij}^-$ for $i = 1, 2, \dots, 6$ and $j = 1, 2$. For the proposed algorithm, in backward phase we want to evaluate

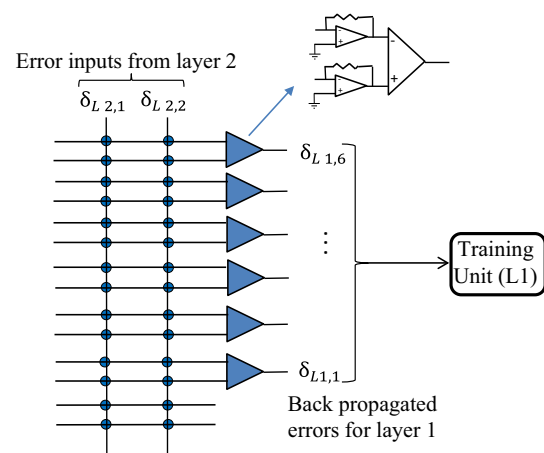


Fig. 4 Schematic of the neural network shown in Fig. 2 for back propagating errors to layer 1

$$\begin{aligned}
 \delta_{L1,i} &= \text{sgn}(\sum_j w_{ij} \delta_{L2,j}) \text{ for } i=1,2,\dots,6 \text{ and } j=1,2. \\
 &= \text{sgn}(\sum_j (\sigma_{ij}^+ - \sigma_{ij}^-) \delta_{L2,j}) \\
 &= \text{sgn}(\sum_j \sigma_{ij}^+ \delta_{L2,j} - \sum_j \sigma_{ij}^- \delta_{L2,j})
 \end{aligned}
 \tag{11}$$

The circuit in Fig. 4 is essentially evaluating the same operations as Eq. (11). Alibart et al. [9] utilized similar design for neuron circuits to implement linear separators. In this approach complexity of back propagation step will be $O(1)$.

Step 3: The training units use the inputs, errors, and intermediate neuron outputs (DP_j) to generate a set of training pulses. The training unit essentially implements Eq. (10). To update a synaptic weight by an amount Δw , the conductance of the memristor connected to the corresponding uncomplemented input will be updated by amount $\Delta w/2$ (where $\Delta w/2 = \eta \times \delta_j \times g_{th}(DP_j) \times x_i$) and the memristor connected with complemented input by amount $-\Delta w/2$.

For design simplicity the error term δ_j , in the training rule, is discretized to a value of either 1, -1 or 0. The key impact of δ_j is to determine the direction (by conductance increase or decrease) of the weight update along with input x_i . The magnitude of the weight update is determined by the values $\eta \times g_{th}(DP_j) \times x_i$. Voltage across a memristor needs to surpass a threshold, V_{th} to change its conductance [15]. For making the desirable changes to memristor conductance values, we need to apply a voltage of appropriate magnitude and polarity for a suitable duration across the memristor. We are determining training pulse magnitude on the basis of neuron input and duration on the basis of $\eta g_{th}(DP_j)$ for a neuron j .

The circuit in Fig. 5 will be used to pulse the memristor crossbar during training when sign of $(x_i \times \delta_j)$ and DP_j are positive. The output of neuron j is essentially producing the sign of DP_j . For each combination of sign of $x_i \delta_j$ and DP_j , inputs to the training pulse generation circuit in Fig. 5(b) are shown in Table 3. The inputs are taken such that during weight increase $V_{wai} > 0$, $V_{wd} = V_{SS}$ for the training period (so that potential across the memristor is greater than V_{th}). During weight decrease, we want to have $V_{wai} < 0$, $V_{wd} = V_{DD}$ for the training period (so that potential across the memristor is less than $-V_{th}$).

The training pulse V_{wdj} is determined by the DP_j value of each neuron. To determine the DP_j value of a neuron we need to access the crossbar column implementing the neuron (the input of the inverter pair in the neuron circuit, Fig. 1(a)). We cannot access the DP_j values of the neurons and update the weights of the neurons at the same time. In this circuit we propose to store the DP_j values in capacitors

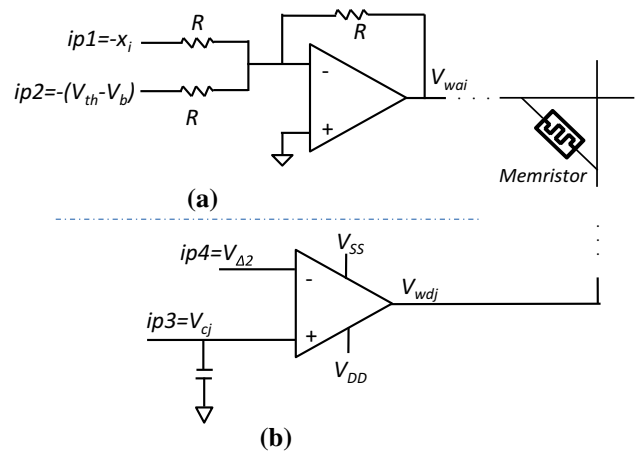


Fig. 5 Training pulse generation module for the proposed training algorithm. Inputs to the circuit are mentioned for the scenario shown in the first row of Table 3. V_{wai} determines training pulse amplitude which is modulated based on input x_i . V_{wdj} determines training pulse duration which is modulated based on $\eta g_{th}(DP_j)$

as V_{cj} (after applying the input to the network) and use it to determine the training pulses of appropriate durations. Because of the inherent normalization operation in the neuron circuit, V_{cj} will be in the range $[-0.5 \text{ V}, 0.5 \text{ V}]$. Assume that $a = 1.9/0.5$ is a scaling factor which will be used to scale up V_{cj} .

The amplitude of V_{wai} can be expressed as $V_{th} - V_b + x_i$ (see Fig. 5(a)). The training circuit utilizes a triangular wave $V_{\Delta 2}$ (see Eq. (12)) to modulate the training pulse duration based on $\eta \times g_{th}(DP_j)$. Duration of $V_{\Delta 2}$, $T_{\Delta 2}$ determines the learning rate. Effective training pulse duration is $T_{\Delta 2}(1 - a|V_{cj}|/2)$ or $T_{\Delta 2}(1 - |DP_j|/2)$ which is consistent with the training rule. Detail description on the training circuit is given in appendix.

$$V_{\Delta 2}(t) = \begin{cases} \frac{4t}{aT_{\Delta 2}} & \text{if } 0 \leq t \leq T_{\Delta 2}/2 \\ \frac{2}{a} - \frac{4}{aT_{\Delta 2}} \left(t - \frac{T_{\Delta 2}}{2} \right) & \text{if } T_{\Delta 2}/2 < t \leq T_{\Delta 2} \\ 0 & \text{otherwise} \end{cases}
 \tag{12}$$

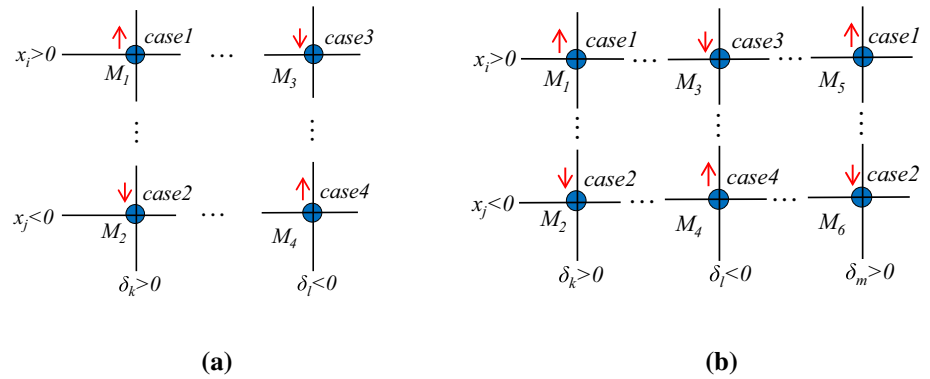
4.4 Writing to memristor crossbars

For the assumed layout of the memristor crossbar, conductance increase requires potential across a memristor (row to column) greater than V_{th} and conductance decrease requires potential less than $-V_{th}$. As a result, in the same row or same column conductance of different memristors cannot be increased and decreased simultaneously. Furthermore, any two weight update operations of the four scenarios in Fig. 6(a) cannot be done simultaneously. For example, if we update M_1 and M_4 simultaneously, M_2 and

Table 3 Inputs to the training module for different scenarios of the proposed training algorithm

Sign of $\delta_j x_i$	Sign of DP _j	Weight update	$ip1$	$ip2$	$ip3$	$ip4$	V_{DD}	V_{SS}
+	+	Increase	$- x_i $	$-(v_{th} - v_b)$	V_c	$V_{\Delta 2}$	0 V	$-V_b$
+	-	Increase	$- x_i $	$-(v_{th} - v_b)$	$-V_{\Delta 2}$	V_c	0 V	$-V_b$
-	+	Decrease	$ x_i $	$-(-v_{th} + v_b)$	$V_{\Delta 2}$	V_c	V_b	0 V
-	-	Decrease	$ x_i $	$-(-v_{th} + v_b)$	V_c	$-V_{\Delta 2}$	V_b	0 V

Fig. 6 **a** Memristor crossbar weight (conductance) update scenarios. Upward arrow indicates increase of conductance and downward arrow indicates decrease of conductance. **b** Updating multiple memristors in a single step. Upward arrow indicates increase of conductance and downward arrow indicates decrease of conductance



M_3 will also be updated to the undesirable direction (increase as opposed to decrease). For each combination of $sign(\delta_j)$ and $sign(x_i)$ shown in Table 4, one exclusive write step is required. This implies, entire memristor crossbar implementing a layer of neurons could be updated in four steps.

Now we will describe the conductance update procedure for all the memristors corresponding to positive inputs ($x_i > 0$) and positive neuron errors (δ_j). This step will update all the memristors belonging to case 1 (which will update multiple memristors residing in different rows and columns simultaneously). In Fig. 6(b) memristors M_1 and M_5 will be updated in this step. This weight update approach is different from the existing approaches which update each crossbar column in two steps [11]. V_{wai} produced by the circuits similar to Fig. 5(a) will be applied to the crossbar rows corresponding to input $x_i > 0$ and 0 V will be applied to the rest of the rows. V_{wdj} produced by the circuits similar to Fig. 5(b) will be applied to the crossbar columns corresponding to error $\delta_j > 0$ and 0 V will be applied to the rest of the columns. Weight update

procedures for the other three cases in Table 3 are similar to the procedure used for case 1.

For a layer of neurons the weight update requires $O(c)$ operations where c is a constant. The training units are only active in the training phase. When evaluating new inputs, only the forward pass will be executed and there will be no power consumed by the training units, comparators, or op-amps used in error back propagation step.

5 Experimental setup

MATLAB (R2014a) and SPICE (LTspice IV) were used to develop a simulation framework for applying the training algorithm to a multi-layer neuron circuit. SPICE was mainly used for detailed analog simulation of the memristor crossbar array and MATLAB was used to simulate the rest of the system. A wire resistance of 5Ω between memristors in the crossbar is considered in these simulations. Each attribute of the input was mapped within $[-V_{read}, V_{read}]$ voltage range. We have performed simulations of the memristor crossbar based neural networks both considering memristor device variation, and stochasticity and without considering device variation, and stochasticity. We assumed a maximum 30% deviation of memristor device response due to device variation and stochasticity. That is, when we want to update a memristor conductance by amount x , corresponding training pulse would update the conductance by a value randomly taken from the interval $[0.7x, 1.3x]$. Table 5 shows the simulation parameters. The resistance of the memristors in the crossbars were randomly initialized between 0.909 and 10 M Ω .

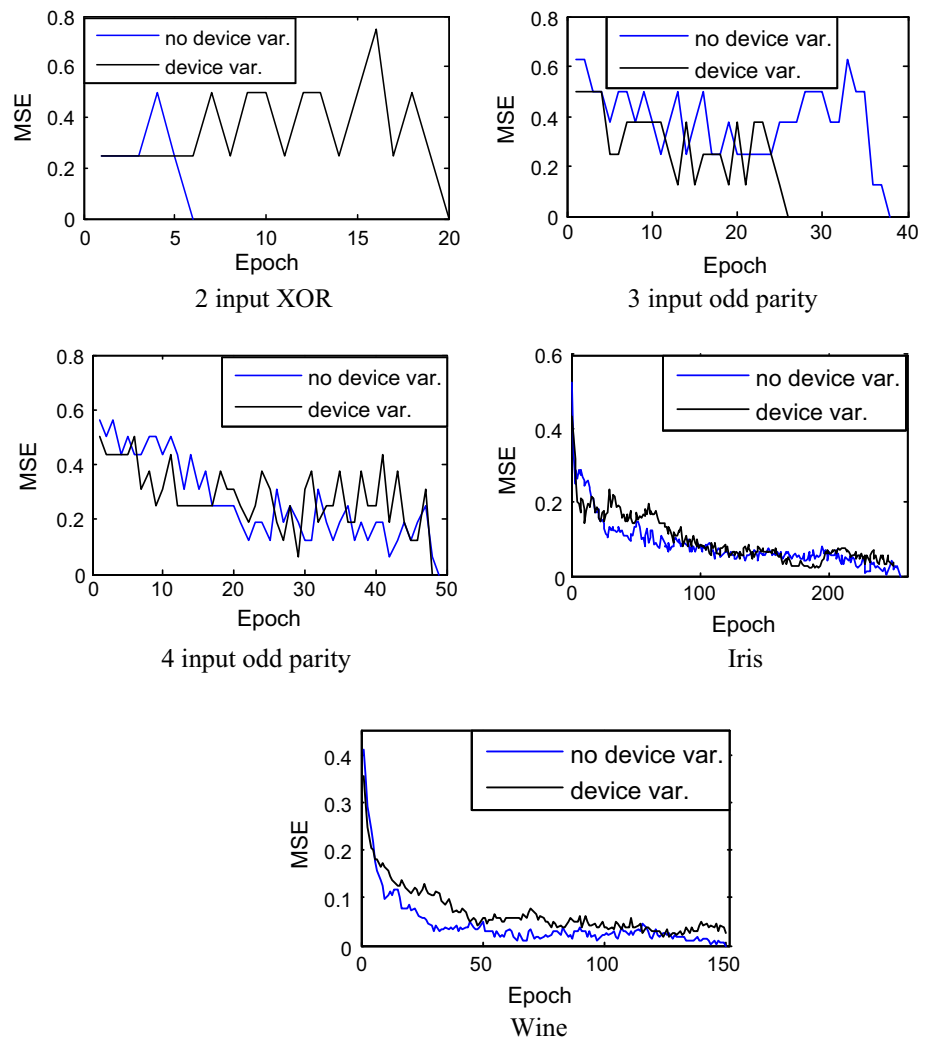
Table 4 Weight update scenarios

	$sign(\delta_j)$	$sign(x_i)$	Weight update
Case 1	+	+	Increase
Case 2	+	-	Decrease
Case 3	-	+	Decrease
Case 4	-	-	Increase

Table 5 Simulation parameters

Memristor R_{ON}	10 k Ω
Memristor R_{OFF}	10 M Ω
Maximum read voltage, V_{read}	0.5 V
Threshold voltage	1.3 V
Memristor switching time for write voltage 2.5 V	20 μ s
Crossbar each wire segment resistance	5 Ω
Maximum deviation of memristor device response due to device variation and stochasticity	30%
Learning rate (maximum training pulse duration)	5–8 ns

Fig. 7 SPICE training results for the proposed algorithm for both cases: without considering memristor device variation, stochasticity (no device var.) and considering device variation, stochasticity (device var.)



Simulation of the memristor device used an accurate model of the device published in [19]. The memristor device simulated in this paper was published in [7]. This device was chosen for its high minimum resistance value and large resistance ratio. According to the data presented in [7] this device has a minimum resistance of 10 k Ω , a resistance ratio of 10^3 , a threshold voltage of about 1.3 V, and the full resistance range of the device can be switched

in 20 μ s by applying 2.5 V across the device. The following parameter values were used in the model in the simulations: $V_p = 1.3$ V, $V_n = 1.3$ V, $A_p = 5800$, $A_n = 5800$, $x_p = 0.9995$, $x_n = 0.9995$, $\alpha_p = 3$, $\alpha_n = 3$, $a_1 = 0.002$, $a_2 = 0.002$, $b = 0.05$, $x_0 = 0.001$.

Table 6 Time required for major operations during training

Operation	Time (ns)
Forward pass	10
Output layer error calculation and store	5
Determine whether nonzero error or not	5
Back propagate error and save error	20
Proposed system entire level 2 weight update	120
Proposed system entire level 1 weight update	100
Single weight update step (BP)	20

6 Results

We have examined the training of the memristor crossbar arrays for five of the datasets in Table 2. We do not show the SPICE training result for the MNIST dataset as the simulation of a memristor crossbar large enough for this application would take very long time. Figure 7 shows the training graphs obtained from the SPICE simulations for different datasets utilizing the training process described in Sect. 4. The results show that the neural networks were able to learn the desired classification applications in both cases: without considering device variation, stochasticity and considering device variation, stochasticity.

We have evaluated the recognition errors on test data for these memristor based neural networks. For Iris, and Wine datasets and the errors were 3.92, and 5.22% respectively when device variation, stochasticity were not considered. Corresponding recognition errors considering device variation, stochasticity were 6.5, and 5.55% respectively. These test errors are close to the errors obtained through software implementations (Sect. 4.2).

Proposed system enables faster weight update operations at low cost hardware implementation of the proposed variant of the BP training algorithm. Table 6 shows the time required for the major operations during training. We assumed, for the traditional BP based system, weight update for multiple layers could be done in parallel. Total training time in the proposed system (*Proposed_time*) and in a state of the art system [11] (*BP_time*) can be expressed as follows.

$$\begin{aligned} \text{Proposed_time} = & \#_of_epoch \times \#_of_sample \times \\ & (\text{forward_pass_time} + \text{backward_pass_time} + \\ & L2_weight_update_time + L1_weight_update_time) \end{aligned}$$

Table 7 Training time comparison

Dataset	Epoch BP	Epoch proposed	BP time (μs)	Proposed time (μs)	Speedup
2 input XOR	30	30	71.4	31.2	2.3
3 input odd parity	40	40	254.4	83.2	3.1
4 input odd parity	50	50	1036	208	5.0
Iris	250	250	43,436.3	6435	6.8
Wine	150	150	40,251.7	4641	8.7

$$\begin{aligned} \text{BP_time} = & \#_of_epoch \times \#_of_sample \times \\ & (\text{forward_pass_time} + \text{backward_pass_time} + \max \\ & \{L2_weight_update_time, L1_weight_update_time\}) \end{aligned}$$

Table 7 compares the training times for the proposed system and a traditional BP algorithm based system [11]. For the examined benchmark datasets, proposed training system was up to $8.7\times$ faster than training using existing approaches. It could be observed that bigger networks provide greater speedup.

7 Conclusion

In this paper we have designed on-chip training systems for memristor based multi-layer neural networks utilizing two memristors per synapse. We have examined training of the memristor based neural networks utilizing the proposed variant of the BP algorithm. For a training instance, weight update operation of an entire memristor crossbar in the proposed system is done in four steps which enables faster training. We have demonstrated successful training of some nonlinearly separable datasets through detailed SPICE simulations which take crossbar wire resistance and sneak-paths into consideration. The proposed training algorithm can train nonlinearly separable functions with a slight loss in accuracy compared to training with the traditional BP algorithm.

Appendix

The inputs to the training circuit in Fig. 5 (as shown in Table 8) are taken such that during weight increase potential across the target memristor ($V_{wai} - V_{wd}$) is greater than V_{th} for the training period. During weight decrease $V_{wai} - V_{wd}$ is less than $-V_{th}$ for the training period. Table 8 shows the potentials of different nodes along with the potential across a target memristor during training. Here V_{cj} is the normalized dot product (DP_j) of inputs and weights stored in the capacitor. Evaluation of V_{wd} is detailed in the following paragraph. Value of V_b is chosen such that $\max\{x_i\} + V_{th} - V_b < V_{th}$, and $-\max\{x_i\} - V_{th} + V_b > -V_{th}$. As a result conductance of

Table 8 Potential of different nodes of the circuit in Fig. 5

Sign of δ_j x_i	Sign of DP_j	Weight update	ip1	ip2	ip3	ip4	V_{DD}	V_{SS}	V_{wd}	V_{wai}	$V_{memristor} = V_{wai} - V_{wd}$	
											For time $T_{\Delta 2}(1 - a V_{cj} /2)$	Rem. time
+	+	Increase	$- x_i $	$-(v_{th} - v_b)$	V_{cj}	$V_{\Delta 2}$	0 V	$-V_b$	$ x_i + V_{th} - V_b$	$ x_i + V_{th}$	$ x_i + V_{th} - V_b$	
+	-	Increase	$- x_i $	$-(v_{th} - v_b)$	$-V_{\Delta 2}$	V_{cj}	0 V	$-V_b$	$ x_i + V_{th} - V_b$	$ x_i + V_{th}$	$ x_i + V_{th} - V_b$	
-	+	Decrease	$ x_i $	$-(v_{th} + v_b)$	$V_{\Delta 2}$	V_{cj}	V_b	0 V	$- x_i - V_{th} + V_b$	$-(x_i + V_{th})$	$- x_i - V_{th} + V_b$	
-	-	Decrease	$ x_i $	$-(v_{th} + v_b)$	V_{cj}	$-V_{\Delta 2}$	V_b	0 V	$- x_i - V_{th} + V_b$	$-(x_i + V_{th})$	$- x_i - V_{th} + V_b$	

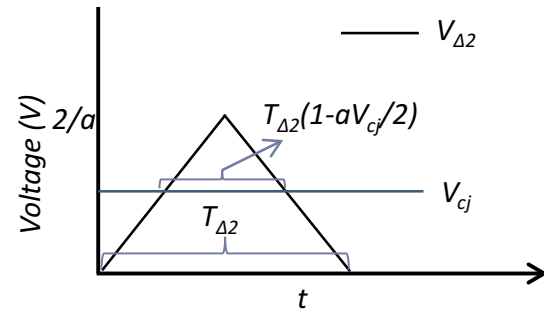


Fig. 8 Plot showing $V_{\Delta 2}$, V_{cj} and training pulse duration $T_{\Delta 2}(1 - aV_{cj}/2)$

the desired memristors are updated only in the training period ($T_{\Delta 2}(1 - a|V_{cj}|/2)$).

$$V_{\Delta 2}(t) = \begin{cases} 4t & \text{if } 0 \leq t \leq T_{\Delta 2}/2 \\ \frac{2}{a} - \frac{4}{aT_{\Delta 2}} \left(t - \frac{T_{\Delta 2}}{2} \right) & \text{if } T_{\Delta 2}/2 < t \leq T_{\Delta 2} \\ 0 & \text{otherwise} \end{cases}$$

Training pulse duration is modulated by the term $\eta \times g_{th}(DP_j)$ in the training rule [Eq. (10)]. This is implemented using the voltage signal $V_{\Delta 2}$ in Fig. 5(b), as triangular wave [Eq. (12)]. In this circuit, when $\delta_j x_i > 0$ and $DP_j > 0$ the comparator gets $V_{DD} = 0$ V and $V_{SS} = -V_b$ (see Table 8). When $V_{cj} > V_{\Delta 2}$ then amplitude of V_{wdj} will be V_{SS} whose value is $-V_b$. V_{cj} is greater than $V_{\Delta 2}$ for the duration $T_{\Delta 2}(1 - aV_{cj}/2)$ where $T_{\Delta 2}$ is the duration of the triangular wave and $a = 1.9/0.5 = 3.8$ (see Fig. 8). The value of $T_{\Delta 2}$ determines the learning rate, η . The time duration $T_{\Delta 2}(1 - aV_{cj}/2)$ is consistent with the term $\eta g_{th}(DP_j)$ in the training rule [Eq. (10)]. Similarly for the other cases in Table 8 potential V_{wd} could be evaluated.

References

1. Taha, T. M., Hasan, R., Yakopcic, C., & McLean, M. R. (2013). Exploring the design space of specialized multicore neural processors. In *IEEE international joint conference on neural networks (IJCNN)*.
2. Belhadj, B., Zheng, A. J. L., Hélot, R., & Temam, O. (2013). Continuous real-world inputs can open up alternative accelerator designs. In *ISCA*.
3. Chua, L. O. (1971). Memristor—The missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5), 507–519.
4. Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. *Nature*, 453, 80–83.
5. Chabi, D., Zhao, W., Querlioz, D., Klein, J.-O. (2011). Robust neural logic block (NLB) based on memristor crossbar array. In *IEEE/ACM international symposium on nanoscale architectures* (pp. 137–143).
6. Taha, T. M., Hasan, R., Yakopcic, C. (2014). Memristor crossbar based multicore neuromorphic processors. In *IEEE international system-on-chip conference (SOCC)* (pp. 383–389).

7. Yu, S., Wu, Y., & Wong, H.-S. P. (2011). Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory. *Applied Physics Letters*, 98, 103514.
8. Medeiros-Ribeiro, G., Perner, F., Carter, R., Abdalla, H., Pickett, M. D., & Williams, R. S. (2011). Lognormal switching times for titanium dioxide bipolar memristors: Origin and resolution. *Nanotechnology*, 22(9), 095702.
9. Alibart, F., Zamanidoost, E., & Strukov, D. B. (2013). Pattern classification by memristive crossbar circuits with ex situ and in situ training. *Nature Communications*, 4, 2072.
10. Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., & Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550), 61–64.
11. Soudry, D., Castro, D. D., Gal, A., Kolodny, A., & Kvatinsky, S. (2015). Memristor-based multilayer neural networks with online gradient descent training. *IEEE Transactions on Neural Networks and Learning Systems*, 26(10), 2408.
12. Li, B., Wang, Y., Wang, Y., Chen, Y., Yang, H. (2014). Training itself: Mixed-signal training acceleration for memristor-based neural network. In *Design automation conference (ASP-DAC), 2014 19th Asia and South Pacific*.
13. Amant, R. S., Yazdanbakhsh, A., Park, J., Thwaites, B., Esmailzadeh, H., Hassibi, A., Ceze, L., & Burger, D. (2014). General-purpose code acceleration with limited-precision analog computation. In *Proceeding of the 41st annual international symposium on computer architecture (ISCA '14)*. IEEE Press, Piscataway (pp. 505–516).
14. Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (2nd ed.). Upper Saddle River: Prentice Hall. (ISBN-13: 978-0137903955).
15. Dong, X., Xu, C., Member, S., Xie, Y., & Jouppi, N. P. (2012). NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 31(7), 994–1007.
16. <https://archive.ics.uci.edu/ml/datasets/Wine>.
17. <https://archive.ics.uci.edu/ml/datasets/Iris>.
18. <http://yann.lecun.com/exdb/mnist/>.
19. Yakopcic, C., Taha, T. M., Subramanyam, G., & Pino, R. E. (2013). Memristor SPICE model and crossbar simulation based on devices with nanosecond switching time. In *IEEE international joint conference on neural networks (IJCNN)*.
20. Zamarreño-Ramos, C., Camuñas-Mesa, L. A., Pérez-Carrasco, J. A., Masquelier, T., Serrano-Gotarredona, T., & Linares-Barranco, B. (2011). On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex. *Frontiers in Neuroscience, Neuromorphic Engineering*, 5(26), 1–22.
21. Chabi, D., Zhao, W., Querlioz, D., & Klein, J.-O. (2011). Robust neural logic block (NLB) based on memristor crossbar array. In *Proceedings of NANOARCH* (pp. 137–143).
22. Starzyk, J. A. (2014). Memristor crossbar architecture for synchronous neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(8), 2390–2401.
23. Sah, M. P., Yang, C., Kim, H., & Chua, L. O. (2012). Memristor circuit for artificial synaptic weighting of pulse inputs. In *IEEE ISCAS*.
24. Adhikari, S. P., Yang, C., Kim, H., & Chua, L. O. (2012). Memristor bridge synapse-based neural network and its learning. *IEEE Transactions on Neural Networks and Learning System*, 23(9), 1426–1435.
25. Zamanidoost, E., Bayat, F. M., Strukov, D., & Kataeva, I. (2015). Manhattan rule training for memristive crossbar circuit pattern classifiers. In *IEEE international joint conference on neural networks*.
26. Zamanidoost, E., Klachko, M., Strukov, D., Kataeva, I. (2015). Low area overhead in situ training approach for memristor-based classifier. In *2015 IEEE/ACM international symposium on nanoscale architectures (NANOARCH)* (pp. 139–142).
27. Kataeva, I., Merrikh Bayat, F., Zamanidoost, E., & Strukov, D. B. (2015). Efficient training algorithms for neural networks based on memristive crossbar circuits. In *IEEE 2015 international joint conference on neural networks (IJCNN)*.



Raqibul Hasan received his B.S. degree in Computer Science and Engineering from the Bangladesh University of Engineering and Technology (BUET) in 2009. He received his M.S. and Ph.D. degrees in Electrical Engineering from the University of Dayton in 2014 and 2016 respectively. His current research includes memristor based low power circuits and systems design.



Tarek M. Taha (S'96-M'03) received the B.S. degree from DePauw University, Greencastle, Indiana, in 1996 and the BSEE, MSEE, and Ph.D. degrees in Electrical Engineering from the Georgia Institute of Technology, Atlanta, in 1996, 1998, and 2002, respectively. He is an Associate Professor of Electrical and Computer Engineering at the University of Dayton. His research interests include cognitive computing architectures, high performance computing, and architectural performance modeling. He received the NSF CAREER Award in 2007. His research is supported by the National Science Foundation, Air Force Office of Scientific Research, the Air Force Research Laboratory, and DARPA. He is a member of the IEEE Computer Society.



Chris Yakopcic (IEEE S'11–M'14) received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from the University of Dayton in 2009, 2011, and 2014 respectively. His current research includes memristor device modeling, analog circuit design with memristor devices, and implementing neuromorphic algorithms on memristor crossbars. In 2013 he was awarded the IEEE / INNS International Joint Conference on Neural Networks best paper award for a paper on memristor device modeling.