Learning Method for Ex-situ Training of Memristor Crossbar based Multi-Layer Neural Network

Anu Bala, Adedotun Adeyemo, Xiaohan Yang and Abusaleh Jabir
School of Engineering, Computing and Mathematics
Oxford Brookes University, Oxford, UK
Email:{15057719,13121679,15058456,ajabir}@brookes.ac.uk

***Abstract*—Memristor is being considered as a game changer for the realization of neuromorphic hardware systems due to its similarity with biological synapse. Recent studies show that memristor crossbar can provide high density and high performance neural network hardware implementation at low power due to its physical layout, nano scale size and low power consumption feature. This paper describes the training method that can be used for the implementation of memristive multi-layer neural network with ex-situ method. We mimic the behavior of memristor crossbar in software training process to achieve more accurate and close computations to hardware. Voltage divider has been used to calculate the dot product in this method. To demonstrate the accuracy and effectiveness of this method, different patterns and non-separable functions using memristor crossbar structures are simulated. The results demonstrate that more accurate computations can be produced using this learning method for ex-situ. It also reduces the learning time of functions.**

***Keywords*-Memristor; neuron circuit; memristor crossbar; neural network; voltage divider.**

I. Introduction

The major obstacles in the neuromorphic systems are to build highly dense, reliable and low power consumption neural systems, which are required for efficient and fast computing [1]. The current CMOS technology is unable to provide high density and high connectivity as well as it consumes more power [2]–[4] and increases the design complexity of neuromorphic systems. Hence, memristor is expected to drive revolution in neuromorphic systems due to its nanoscale size, non-volatility, high density, low power consumption as well as compatibility with CMOS technology.

Memristor is the fourth basic fundamental element with variable resistance that was proposed by Leon Chua in 1971 [5]. The proposed theory predicts the relationship between the charge and magnetic flux. HP labs fabricated the first physical memristor model in 2008 [6] and it reveals the same behavior as predicted by Leon Chua in 1971. Applying voltage pulses at its terminals can alter the resistance of a memristor device. Owing to the similarity between memristor devices and biological synapses, more research interests have been seen in the neuromorphic domain. Memristor devices are being utilized as synapses in the implementation of neural systems.

Memristor can be used in memory [7] as well as logic design [8], [9]due to its retaining proficiency. Moreover, it can also be used in analog circuits [10] and image processing for storage and memory organization due to its nano scale size and low power consumption features [11]. A memristor device presented in [1] demonstrated that it could be used as synapse in neuromorphic applications and higher synaptic density could be achieved by using high-density memristor crossbar grid due to its physical layout. Numerous neuromorphic systems are developed by using memristive crossbar circuits in [12] [13] which illustrates the potential of the memristive crossbar structure in massive neural network implementation. Hence, memristors can be effectively used to build neural networks with a large number of neurons and synaptic connections.

The training methods are required to train any neural network. The weight value of each neuron in the neural network is calculated and updated using the training methods. Two basic training methods are used to train the networks called ex-situ and in-situ. During ex-situ training process, the training is provided in software; final weights are calculated and imported into the hardware. In the in-situ method, the weight value of each memristor devices has been updated after each iteration during training process in hardware. The key benefit of ex-situ method is that any learning algorithm can be used for training, which is difficult in the in-situ method. However, to achieve accuracy in ex-situ is difficult due to lack of efficient mapping schemes and device variability.

Recent studies show that different mapping schemes and neuromorphic circuits are proposed for ex-situ to achieve more accurate computations between hardware and software [12] [14]. This paper presents the modified training method for ex-situ. The memristor crossbar behavior has been studied deeply from literature and we tried to model the same behavior of crossbar in software training. We calculated the dot product using voltage divider so that software computations could be matched more closely to hardware computations and more accuracy can be achieved. Thus, the network trained in software mimic the memristor crossbar in hardware.

The rest of the paper is organized as follows: Section II describes the neuron circuit, proposed learning method and the experimental setup for the implementation of memristive neural network based on ex-situ training method. Section III describes memristor crossbar based single-layer and multi-layer neural networks implementation using proposed learning method introduced in section II. It also shows the experimental results and comparison between the training method using simple dot product and our proposed method. The paper is concluded in section IV.

II. Memristor based Neural Network Circuit Design and Learning

A. Neuron Circuit

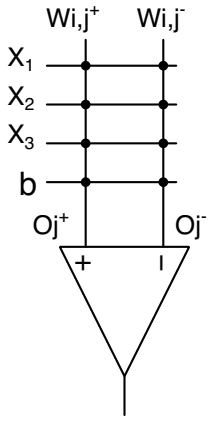


Fig. 1: Memristor based Neuron Circuit

The schematic in Fig. 1 displays memristor based neuron circuit, which is used to train and implement ex-situ neural network. The training has been provided in software and final weight values have been written in hardware. Conductance values are considered as weight values in the memristor based neural network circuits. So the weights are updated according to the conductances of memristor in memristive neural networks. The 0T1M approach is used in the circuit as shown in fig.1 as 0T1M memristor crossbars are much denser and smaller in area than 1T1M crossbar structures.

In this circuit, the rows represent the input values and columns the outputs. X_1, X_2 and X_3 are inputs and b is considered as bias in this structure. Memristor acts as synapse and interconnects each row with the column. Two memristors are used per synapse to present either positive or negative effect on total weight values. Using the following equation, the value of W in the memristor crossbar array is always calculated.

$$W = W_{i,j^+} - W_{i,j^-} \tag{1}$$

The weight value in the memristor crossbar is obtained by subtracting W_{i,j^-} from W_{i,j^+}.

All the inputs along with memristors of each column are connected to a comparator as shown in fig.1. If the output of the column on the positive side of the comparator is greater than the column output on the negative side of the comparator, then the pair of memristors used per synapse represents a positive weight and vice versa. The output of the comparator represents the output of each neuron. The comparator gives an output as two discrete values as it behaves as ideal threshold function.

The Dot product is the most important part in any neural network algorithm. The output of any neuron depends on the dot product and activation function in neural networks. The dot product of any neural network in software is represented by Eq. (2). However, any simple memristor crossbar array provides the output as shown in Eq. (3).

$$Vout_j = \sum\nolimits_i X_i W_{i,j} \tag{2}$$

Here, X_i represents the input values and $W_{j,i}$ represents the weight values in software neural networks.

$$Vout_j = \frac{\sum_i Vin_i C_{i,j}}{\sum_i C_{i,j}} \tag{3}$$

Vin_i and $C_{i,j}$ represnt voltage and conductance values corresponding to each input i.

The output at each column in memristor crossbar is calculated as Eq. (3). It represents a voltage divider between the memristor conductance values as conductance is the reciprocal of resistance.

B. Modified Learning Method for Ex-$situ$ Training

In the proposed training method, the dot product at each column has been calculated using the voltage divider as our training process mimics the memristor crossbar in hardware. The simple memristor crossbar structure uses voltage divider between the resistances connected in series and parallel to provide the output at each column as described in the previous section. Hence, all the networks have been trained using the voltage divider instead of using simple dot product. The arctan function is used as an activation function. Two weight values per synapse are used in the learning process. The back propagation learning algorithm [15] has been used to train the networks with modified dot product and weight update equations. As it is the most powerful learning algorithm, however, its implementation in hardware is difficult. Following are the steps of the learning algorithm that have been used to train the networks in software.

1) Initialize the weights with low random conductance values.
2) Calculate the dot product at each column of the output layer and the hidden layer as the networks are trained in crossbar fashion. The output value of each column is calculated using the following equations as it represents the voltage divider. Then apply the activation function.

$$\begin{aligned} O_j &= \frac{\sum_i X_i W_{i,j}}{\sum_i W_{i,j}} \\ Y_j &= f(O_j) \end{aligned} \tag{4}$$

where,

$$O_j = O_j^+ - O_j^- \tag{5}$$

Here, i and j represent input and hidden-layer nodes. Eq. (4) shows the dot product and activation function of each neuron at hidden-layer.
The following Eq. (6) represents the output and activation function at hidden-layer.

$$O_o = \frac{\sum_j Y_j W_{j,o}}{\sum_j W_{j,o}} \qquad Y_o = f(O_o) \tag{6}$$

where,

$$O_o = O_o^+ - O_o^- \tag{7}$$

The output of the hidden-layer is considered as input to the output layer as shown in Eq. (6). $W_{j,o}$ is the weight value between hidden-layer and output layer.

3) Calculate the error at output layer and hidden-layer using the following equations:

$$E_o = T_o - Y_o \tag{8}$$

Here, E_o is the error at output layer. T_o and Y_o represent the target and actual outputs and backpropagate the error to the previous layer. E_o represents Error function having values either +1, -1, or 0 (corresponds to weight increase, decrease or no change respectively).

$$E_h = \sum\nolimits_j E_o W_{j,o} \tag{9}$$

Here, E_h is the error at hidden-layer. $W_{j,o}$ represents the weight values of synapse connecting neuron o with previous layer neuron j and E_o is the error which is back propagated from the output layer.

4) Update the weights using following equations at each layer:

$$\Delta W_{j,o} = \eta E_o \frac{1}{1+O_o^2} X_{j,o} \tag{10}$$

$$\Delta W_{i,j} = \eta E_h \frac{1}{1+O_j^2} X_{i,j} \tag{11}$$

$$W^+_{j,o,new} = W^+_{j,o,old} + \Delta W_{j,o} \tag{12}$$

$$W^-_{j,o,new} = W^-_{j,o,old} - \Delta W_{j,o} \tag{13}$$

$$W^+_{i,j,new} = W^+_{i,j,old} + \Delta W_{i,j} \tag{14}$$

$$W^-_{i,j,new} = W^-_{i,j,old} - \Delta W_{i,j} \tag{15}$$

Here, η is a constant learning function. The above mentioned Eq. (10) and Eq. (11) determine the total amount of weight change at the output layer and the hidden-layer. Eq. (12) and Eq. (13) are used to update the weights connected with the output layer and Eq. (14) and Eq. (15) are used to update the weights of the hidden-layer.

5) All the steps from step 2 are repeated for each data pattern until the error becomes zero.

C. Experimental Setup

A single and multi-layer layer neural networks based on memristor have been trained and implemented to perform linear, non-linear separable functions and pattern classifiers. All the networks have been trained in software using the proposed training method and then, final weights are written into the memristor crossbar neural circuits in hardware. The C++ environment is used for training and SPICE is used for hardware simulation of these networks.

The memristor model published in [16] [17] has been utilized for the implementation of memristor crossbar in hardware due to its high resistance ratio (R_{off}/R_{on}=10^6). The maximum and minimum resistance values used for this simulation are 125MΩ and 125KΩ. 0T1M crossbar structure is simulated in SPICE to implement these networks. 0T1M memristor crossbar structures are significantly denser and consume less area than 1T1M structures. Write and read operations have been performed on memristor crossbar in SPICE to program the final pre-determined weights. The resistance of each memristor is set according to the target resistance calculated during training in software.

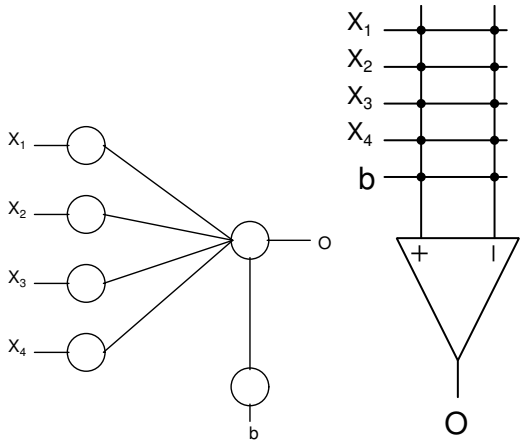
III. Memristor based Neural Network Implementation and Results

This section explains the implementation and results of a single layer and multi-layer neural networks. We have implemented different non-linear separable functions and pattern classifiers using memristive crossbar circuits. First, the networks have been trained in software using general dot product and proposed dot product calculations in the learning process. Then, the final weight values as conductance values of memristors that are calculated in software have been directly updated in memristor based crossbar structure in hadware. The networks have been tested and the experimental results produced from both methods are compared.

A. Single Layer Network

This section illustrates the classification of a linear function and pattern classifier using single layer neural network.

1) Four Bit Linear Function: The Fig.2a shows a single layer network and Fig.2b displays the corresponding memristive crossbar circuit for four input linear function. The rows represent the input values and columns the outputs. X_1, X_2, X_3 and X_4 are inputs and b is considered as bias in a memristive crossbar. Memristor acts as a synapse and interconnects each row with the column. All the inputs along with the pair of memristors are connected to a comparator. The comparator output is the output of each neuron at each layer. The Fig.3 represents the number of epochs required for both the methods to learn linear function. The proposed method required fewer epochs to learn the function as compared to simple dot product method as shown in Fig.3. After hardware implementation of both methods, the general dot product method shows 6.25% error rate during testing. However, our method shows zero error as shown in Table I.



(a) (b)

Fig. 2: (a) Single-Layer Network (b) Memristor Neuron Crossbar Circuit for Linear Function

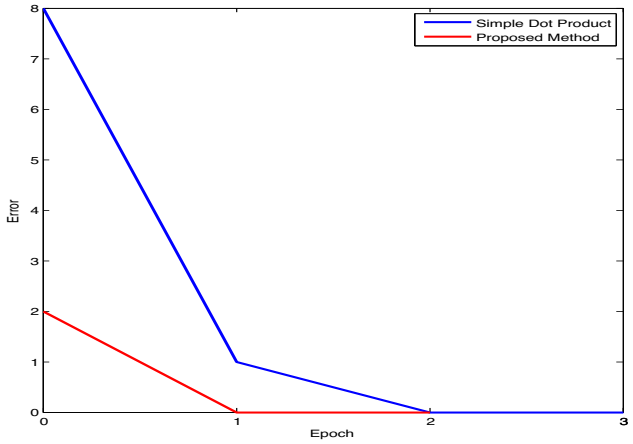


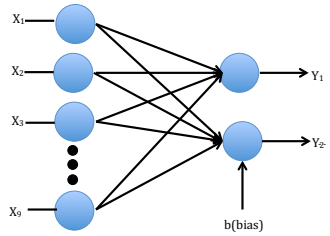
Fig. 3: Training error in each epoch for four input linear function

2) Pattern Classifier: This section presents pattern classification for 3×3 binary images using a single-layer network. The network has been trained through perceptron learning algorithm with modified dot product and weight update equations. Fig. 11 displays 3×3 binary pixel image and single layer perceptron network for pattern classification. The network consisting of nine inputs, one bias and two outputs. The value of bias is fixed and considered as 1, whereas the values of inputs and outputs are 1 or 0. The memristor crossbar for pattern classification contains 10×4 memristors in total including bias as two memristors per synapse.

The set of patterns considered for classification in Fig. 5a and Fig. 5b represents the alphabets 'X' and 'T' and their noisy versions. First patterns in Fig. 5a and Fig. 5b are ideal patterns for letters 'X' and 'T' and rest are noisy patterns. The grey and white pixels of images are represented by 1 and 0 respectively and the output value for pattern 'X' is considered as 1 and 'T' is 0. The training set contained 50 patterns and some of these patterns are repeated.

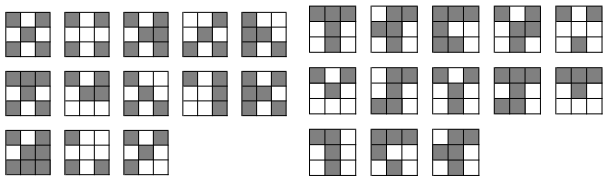
The results in Fig. 6 show the number of epochs required

X_1	X_2	X_3
X_4	X_5	X_6
X_7	X_8	X_9



(a) (b)

Fig. 4: (a) 3×3 Binary Image (b) Single-Layer Network for Pattern Classification



(a) (b)

Fig. 5: (a) Set of input patterns for 'X' (b) Set of input patterns for 'T'

for average error to reach zero. The network is learned only in 3 epochs by proposed method. However, simple dot product method takes two more epochs to learn. During the implementation and testing in hardware, our method shows 100% accuracy than another method as mentioned in Table I.

B. Multi-Layer Crossbar Network Implementation

A multi-layer neural network is required for the classification of complex non-linear separable functions. We implemented the multi-layer neural networks with one and two hidden layers to classify three-bit parity functions and full adder function.

1) Two-Layer Network:

The two-layer neural network is utilized to perform three-bit parity and full adder functions.

a) Three Bit Parity Function: The network in Fig.7a displays a two-layer neural network for three input parity function. The network contains one hidden-layer consisting of four hidden neurons. The Fig.7b displays the memristor crossbar neuron circuit. In this circuit, first layer consisting of 4×8 memristor crossbar and the output layer consists of 5×2 including bias. The Fig.8 represents the results of training with both methods. The network with proposed method learns more quickly as compared to another method. Proposed method requires less than 10 epoch to train the network whereas simple dot product network learns three input parity function in almost 70 epochs and its error rate is more than the proposed method while implementing and testing in hardware as shown in Table I.

b) Full adder Function: The full adder function has also been trained and implemented using the two-layer neural network. The two-layer network and its corresponding memristor crossbar circuit for full adder function are displayed in Fig.9a and Fig.9b. This network contains 3-input neurons, four hidden

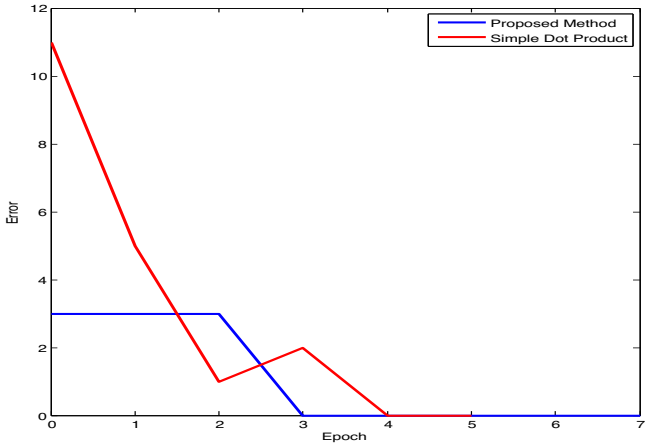


Fig. 6: Number of epochs required for average error to reach zero

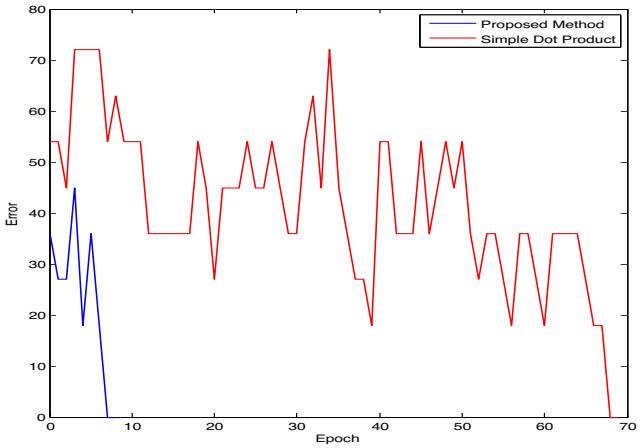
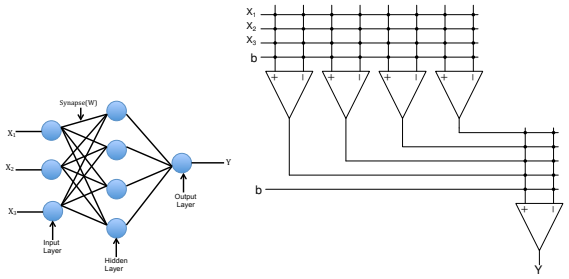
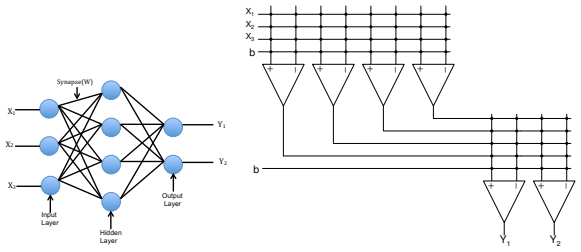


Fig. 8: Epoch required to learn three bit parity function on two-layer network



(a) (b)

Fig. 7: (a) Two-Layer Network and (b) Memristor based Neuron Crossbar Circuit for three-bit parity function



(a) (b)

Fig. 9: (a) Two-Layer Network and (b) Memristor Crossbar Circuit for Full adder Function

neurons and two output neurons. One output is for sum and other is for carry in full adder function. In memristive crossbar, the first layer and output layer have been implemented using 4×8 and 5×4 memristor crossbars respectively. The error reaches to zero after 40 epochs using proposed method and it required more than 200 epochs to learn with simple dot product method. Table I shows 50% error rate during testing and implementation in hardware.

2) Three-Layer Network Simulation:

This section describes the implementation of the three-layer neural network to perform three-bit parity function. A three-layer neural network is shown in Fig. 11a for three input parity function. The network consists of 3 input $\times$ 4 hidden-1 $\times$ 2 hidden-2 $\times$ 1 output nodes to classify 3-bit parity function. Hidden-layer1 consisting of four hidden neurons and hidden-layer2 contains two hidden neurons as shown in Fig. 11a. We used this three layer network to perform the non-linear separable function. The memristive crossbar circuit corresponding to this network which is implemented in hardware is shown in Fig. 11b. In hardware, the first layer is implemented using 4×8 crossbar, the second layer is implemented using 5×4 crossbar and the final layer contains 3×2 crossbar.

Fig. 12 shows the number of epochs required to learn three-bit parity function with both methods. The simple dot product learning method takes more epochs to learn as shown Fig. 12. In the proposed method, after nine epochs, an error becomes zero and the network is learned three-bit parity function successfully. The proposed method also shows the better result than another method during simulation and testing of the memristive crossbar circuit in hardware as shown in Table I.

Our simulation results have successfully demonstrated that the modified learning method is capable of train complex non-linear functions successfully using multi-layer crossbar networks. The experimental results also show that it provides the training faster than using simple dot product in learning algorithm. Moreover, it provides more accuracy while using it with ex-situ method. We achieved zero error rate with the implementation of these small networks using this method. Our aim is to mimic the behavior of memristor crossbar in software training to obtain more close computations to hardware in ex-situ. It also works well with the back propagation learning algorithm. Hence, more accuracy could be achieved for more massive networks using ex-situ process by finding the ways to produce more accurate computations.

TABLE I: Comparison of dot product methods used in software training

		Error rate during testing in hardware	
Function	**Network Configuration**	**Simple Dot Product(%)**	**Proposed Method(%)**
Four-bit-AND	4-1	6.25	0
Pattern Classifier	9-2	2	0
Three-bit-parity	3-4-1	37.5	0
Fulladder	3-4-2	50	0
Three-bit-parity	3-4-2-1	37.5	0

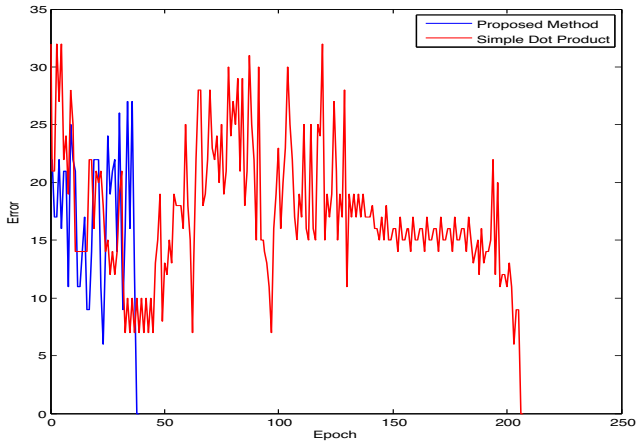
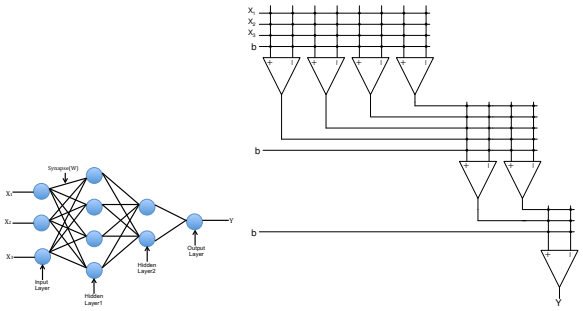


Fig. 10: Error during training process of fulladder function



(a) (b)

Fig. 11: (a) Three-Layer Network and (b) Memristive Neuron Crossbar Circuit for Parity Function

IV. Conclusion

In this paper, a modified learning method has been proposed. We used memristive crossbar architecture to simulate and train a single layer and multi-layer networks by using C++ and SPICE environment. The networks trained in software mimic the memristor crossbar in hardware. Therefore, voltage divider has been used to calculate dot product in the learning process.The aim of this method is to reduce the error rate while implementing networks in hardware with ex-situ training methods. It also reduces learning time in software. The simulation results have successfully demonstrated the classification of linear and non-linearly separable problems and provide more accuracy during the implementation of ex-situ method.

References

[1] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[2] G. Indiveri and T. K. Horiuchi, "Frontiers in neuromorphic engineering," *Frontiers in neuroscience*, vol. 5, p. 118, 2011.

[3] J. Hasler and H. B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers in neuroscience*, vol. 7, p. 118, 2013.

[4] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[5] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

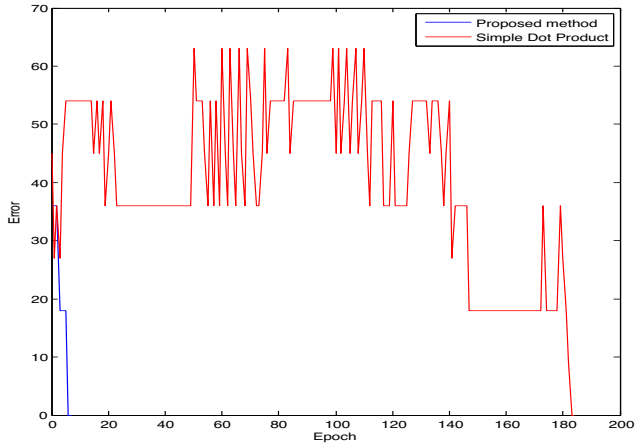


Fig. 12: Error during training process of three-bit parity function on three layer network

[6] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[7] Y. Ho, G. M. Huang, and P. Li, "Nonvolatile memristor memory: device characteristics and design implications," in *ACM International Conference on Computer-Aided Design*, 2009, pp. 485–490.

[8] M. Soltiz, D. Kudithipudi, C. Merkel, G. S. Rose, and R. E. Pino, "Memristor-based neural logic blocks for nonlinearly separable functions," *IEEE Transactions on computers*, vol. 62, no. 8, pp. 1597–1606, 2013.

[9] X. Yang, A. A. Adeyemo, A. Jabir, and J. Mathew, "High-performance single-cycle memristive multifunction logic architecture," *Electronics Letters*, vol. 52, no. 11, pp. 906–907, 2016.

[10] Y. V. Pershin and M. Di Ventra, "Practical approach to programmable analog circuits with memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 8, pp. 1857–1864, 2010.

[11] J. Zhou, Y. Tang, J. Wu, and X. Yi, "Image segmentation with threshold based on memristors," in *IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)*, 2013, pp. 41–44.

[12] C. Yakopcic, R. Hasan, and T. M. Taha, "Memristor based neuromorphic circuit for ex-situ training of multi-layer neural network algorithms," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–7.

[13] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[14] M. S. Tarkov, "Mapping neural network computations onto memristor crossbar," in *2015 International Siberian Conference on Control and Communications (SIBCON)*, May 2015, pp. 1–4.

[15] S. Russell and P. Norvig, "Ai a modern approach," *Learning*, vol. 2, no. 3, p. 4, 2005.

[16] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE Electron Device Letters*, vol. 32, no. 10, pp. 1436–1438, Oct 2011.

[17] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug 2013, pp. 1–7.