2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

# Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors

Raqibul Hasan and Tarek M. Taha

***Abstract*—Recent studies have shown that memristor crossbar based neuromorphic hardware enables high performance implementations of neural networks at low power and in low chip area. This paper presents circuits to train a cascaded set of memristor crossbars representing a multi-layered neural network. The circuits presented implement back-propagation training and would enable on-chip training of memristor crossbars. On-chip training of memristor crossbars can be necessary to overcome the effect of device variability and alternate current paths within crossbars being used as neural networks. We model the memristor crossbars in SPICE in order capture alternate current paths and the impact of wire resistance. Our design can be scaled to multiple neural layers and multiple output neurons. We demonstrate the training of up to three layered neural networks evaluating non-linearly separable functions through detailed SPICE simulations. This is the first study in the literature we have seen that examines the implementation of back-propagation based training of memristor crossbar circuits. The impact of this work would be to enable the design of highly energy efficient and compact neuromorphic processing systems that can be trained to implement large deep networks (such as deep belief networks).**

***Key words:*** Neural networks; memristor crossbars; neuromorphic architectures.

## I. INTRODUCTION

Embedded neuromorphic processing systems have significant advantages to offer, including the ability of solve complex problems and the potential to consume very low power and area. Several studies [1,2] have shown that specialized neuromorphic architectures consume very low processing power. Taha et al. [1] have compared the performance and power of several processing options for specialized neuromorphic systems. They have shown that memristor [1] crossbar based architectures can enable over 5 orders of power reduction over Intel Xeon processors neural network evaluations, while also reducing chip area dramatically.

With the reliability and power consumption for general purpose computers becoming increasing problematic [3], recent studies have started examining the potential of mapping of applications traditionally computed on general purpose computers, to neural network form. The key objective is to take advantage of the low power consumption of specialized neuromorphic hardware and the inherent fault tolerance of neural algorithms. Chen et al. [4] have shown that Recognition, Mining, and Synthesis (RMS) applications (described by Intel as the key future application drivers [5]) can be represented as neural networks. They make the case that neural network accelerators can have broad applications. Esmaeilzadeh et al. [6] show that several key application kernels (such as FFT and JPEG) can be approximated using neural networks and made the case for specialized neural network accelerators on general purpose CPUs.

Memristors [7] have received significant attention as a potential building block for neuromorphic systems [8,9]. The recent physical realization of the memristor [10] has produced a nanoscale non-volatile device with a large varying resistance range. Just as chemical pulses alter synaptic weights in brain tissue, voltage pulses can be applied to memristors to alter their conductivity.

Physical memristors can be laid out in a high density grid known as a crossbar [11]. Using this layout, memristors have the potential to be fabricated with a synaptic density greater than that of brain tissue [12]. As shown in [1], using these memristor crossbars allow high density, extreme low-power, neuromorphic hardware that is capable of performing many multiply-add operations in parallel in the analog domain. Beside their area and computational efficiency, the non-volatile nature of memristors can reduce the static power consumption of these systems significantly.

To work around the device variability that may be present in a memristor crossbar, it may be necessary to develop on-chip training hardware for memristor crossbar based neuromorphic processors. One of the most commonly used techniques to train neural networks is the back-propagation algorithm [13]. In this paper we develop circuits that can enable back-propagation based training of neural networks implemented through memristor crossbars. Our design can be scaled to multiple neural layers and multiple output neurons. We demonstrate the training of up to three layered neural networks evaluating non-linearly separable functions through detailed SPICE simulations of the memristor crossbar circuits. The use of SPICE simulations is essential to capture the impact of alternate current paths that exist in these circuits.

This is the first study in the literature we have seen that examines the implementation of back-propagation based training of memristor crossbar circuits. The impact of this work would be to enable the design of highly energy

This work was supported through an NSF CAREER Award.

R. Hasan is with the University of Dayton, Dayton, OH 45469, USA. (email: hasanm1@udayton.edu).

T. M. Taha is with the University of Dayton, Dayton, OH 45469, USA (phone: 937-229-3119; email: tarek.taha@udayton.edu).

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

efficient and compact neuromorphic processing systems that can be trained to implement large deep networks (such as deep belief networks [14]). The rest of the paper is organized as follows: section II describes related works in the area. Section III examines memristor based neural network implementations. Sections IV and V demonstrate experimental setup and results respectively. Finally, section VI summarizes our work.

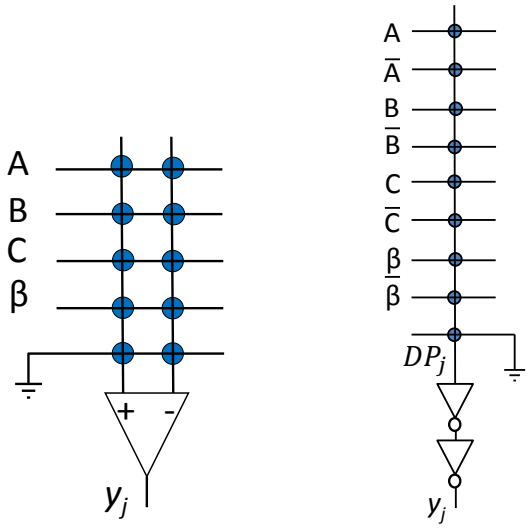
## II. Related Work

Specialized architectures can significantly reduce the power consumption for neural network applications and yet provide high performance [15-18]. IBM and Cornell [17,18] recently developed a fully digital synaptic core in 45 nm SOI technology. The system utilizes asynchronous logic to reduce power consumption. Each core models 256 integrate and fire neurons, with 1024 pre-synaptic inputs per neuron. Each synapse is represented through a pair of bits in a 1024×256 bit SRAM crossbar memory that allows selection of up to three synaptic values (stored in a separate set of registers for each neuron). Rows within the SRAM array are activated depending on which input pre-synaptic connections fired. They proposed that multiple cores would communicate asynchronously using an address-event representation (AER) for spikes. In their study, training was carried out offline as the system is digital.

Belhadj et al. [2] proposed a multicore architecture for spiking neural applications and examined the response of some signal processing applications on that system. They stored synaptic weights in digital form and used digital to analog converters to process the neurons.

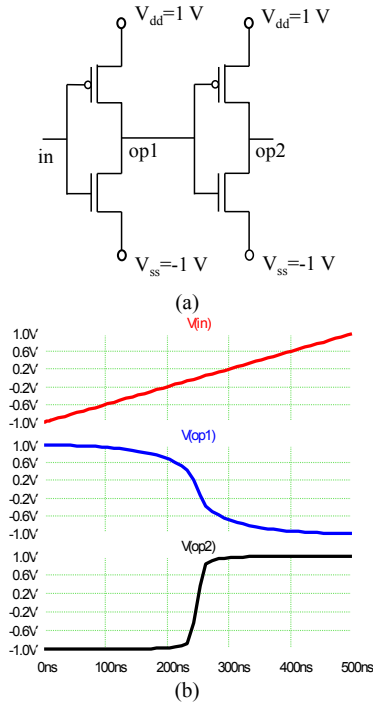
Several studies have examined MOS transistor based analog neuron circuits [17-22]. The Neurogrid project [20,22] aims to model cortical circuits using spiking neuron circuits. They used local analog wiring to minimize the need for digitization of on-chip communications.

Zamarreño-Ramos et al. [23] proposed the use of a memristor grid to implement a highly dense neural network that can be used in visual image processing. Work has been completed that shows how memristors can be used in neural logic blocks in a Field Trainable Neural Array (FTNA) [24]. Each neural block contained a memristor crossbar array and CMOS learning cells to program the memristive weights. Chabi et al. [25] examined the implementation of linearly separable Boolean functions in Neural Logic Blocks (NLB) without taking sneak paths into consideration. They focused on robustness studies of NLB by using probabilistic predictive models of defects. Alibart et al. [26], demonstrated pattern classification using a single-layer perceptron network implemented with a memristive crossbar circuit and trained using the perceptron learning rule by ex situ and in situ methods. They did not study non-linearly separable problems.




(a) (b)

Fig. 1. Two implementation of memristor-based neuron circuit.




(a)

(b)

Fig. 2. (a) CMOS inverter and (b) inverter transfer curve. V(op2) is the neuron output and V(op1) is the inverted neuron output.

## III. Memristor Crossbar Based Neural Network Implementation

### A. Neuron Circuit

Two possible approaches for implementing a neuron circuit using memristors are shown in Fig 1. The neurons shown have three inputs and a bias input. Each synapse is represented by a pair of memristors as shown. Both types of circuits are used in this study.

In the first neuron circuit (Fig. 1(a)), each input is connected through a pair of memristors to a comparator. If the conductance of the memristor on the positive input to the

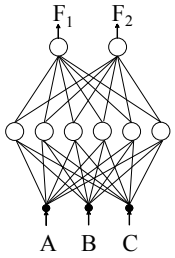



Fig. 3. Two layer network for learning three input odd parity function.

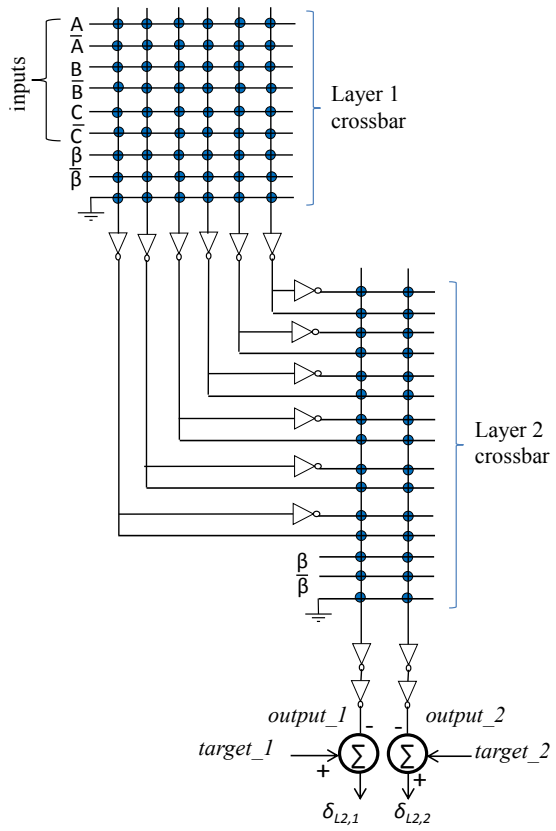



Fig. 4. Schematic of the neural network shown in Fig. 3 for forward pass.

comparator is higher than the other memristor, then the pair of memristors represents a positive synaptic weight. The inverse represents a negative synaptic weight. The output of the comparator represents the neuron output. In addition to the three inputs A, B, and C, a bias input is necessary, along with a path to ground.

In the second neuron circuit (Fig. 1(b)), each input signal and its complemented form (same magnitude but opposite polarity) are applied to a column of memristors. If the conductance of the memristor connected with an input signal is greater that the conductance of the memristor connected with the corresponding inverted signal, then the pair of memristors represents a positive weight (otherwise they represent a negative weight). The output of inverter pair represents the neuron output.

Assume that for a neuron, $x_i$ are the inputs and $w_{j,i}$ are the corresponding synaptic weights. In this case,

$$DP_j = \sum_i x_i w_{j,i} \quad (1)$$

and the neuron output is

$$y_j = f(DP_j) \quad (2)$$

where $f$ is the activation function. In a multi-layer feed forward neural network, a differentiable activation function (e.g. $tan^{-1}(x)$) is desired. We are approximating the output of the neuron in Fig 1a as +1 or -1 through a comparator.

In Fig 1(b), we are approximating the activation function using a pair of CMOS inverters with the power rails having $V_{dd}$=1V and $V_{ss}$= -1V. Fig. 2(a) shows the inverter circuit and the transfer curve is shown in Fig. 2(b), where *V(op2)* is the neuron output and *V(op1)* is the inverted neuron output. This approach provides a very efficient way of implementing the activation function in terms of power, speed and circuit components. Moreover it provides a continuous neuron output, whereas the use of a comparator gives an output of two discrete values (1 or -1).

### *B. Memristor Crossbar Based Neural Network*

Fig. 3 shows a simple two layer feed forward neural network with three inputs, two outputs, and six hidden layer neurons. Fig. 4 shows a memristor crossbar based circuit to evaluate the outputs of the neural network in Fig. 3. There are two memristor crossbars in this circuit, each representing a layer of neurons. Each crossbar utilizes the neuron circuit shown in Fig. 1(b).

A neural network layer with $n$ neurons and $m$ inputs per neuron could be implemented using a $(2m+3)\times n$ memristor crossbar ($2m$ memristors per column from the inputs, and 3 from the bias and path to ground). This provides a very compact and fast way of implementing a layer of neurons. When the inputs are applied to the crossbar, the intermediate outputs $DP_j$ appear at the end of the crossbar columns almost instantly. Thus an entire crossbar can be processed in parallel within a cycle.

In Fig. 4, the first layer of neurons is implemented using a 9×6 crossbar. The second layer of neurons is implemented using a 15×2 memristor crossbar, where 12 of the inputs are coming from the 6 outputs of the first crossbar.

### *C. The Back-propagation Algorithm*

In this paper we are utilizing the Back-propagation algorithm [13] for updating memristor crossbar resistances to implement non-linearly separable functions. The algorithm for training a memristor cross bar based on the back-propagation is stated as:

1) Initialize the memristors with high random resistances.
2) For each input pattern *x:*
   3) Apply the input pattern $x$ to crossbar circuit and evaluate hidden neuron and output neuron values.
   4) For output layer neurons, calculate the error, $\delta_j$, between the neuron output ($F_j$) and the target output ($D_j$):

$$\delta_j = D_j - F_j. \quad (3)$$

5) Back propagate the error.
$$\delta_j = \sum_k \delta_k w_{k,j} \quad (4)$$
where neuron $k$ has connection with previous layer neuron $j$.

6) Determine the amount, $\Delta w$, that each memristor's conductance should be changed ($\eta$ is the learning rate):
$$\Delta w_{j,i} = \eta \times \delta_j \times \frac{1}{1+DP_j^2} \times x_{j,i}. \quad (5)$$

7) Apply write pulses to the crossbar with pulse widths proportional to $\Delta w_{j,i}$ to update the memristor conductances.

8) If the error does not converge to a sufficiently small value, go to step 2.

### D. Circuit implementation of back-propagation training

The circuit implementation of back-propagation training for the neural network in Fig. 3 can be broken down into four major steps:

1. Apply inputs to layer 1 and record layer 2 neuron output errors.
2. Back-propagate layer 2 errors through second layer weights and record layer 1 errors.
3. Update layer 2 synaptic weights based in part on layer 2 errors.
4. Update layer 1 synaptic weights based in part on layer 1 errors.

The circuit implementations of these four steps are detailed below:

*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. 4. The terms $\delta_{L2,1}$ and $\delta_{L2,2}$ are the error terms and are based on the difference between the observed outputs and the expected outputs. These values are generated using a comparator that provides a discretized error value of +1 or -1. Thus these errors can easily be recorded in binary form for later use. More complex circuitry (such as op amps) can be utilized to give a higher resolution on the error, but will be more expensive in terms of power and area. We will explore the impact of using these circuits as future work.

*Step 2:* The layer 2 errors ($\delta_{L2,1}$ and $\delta_{L2,2}$) are applied to the layer 2 weights as shown in Fig. 5 to generate the layer 1 errors ($\delta_{L1,1}$, $\delta_{L1,2}$, etc). The memristor crossbar in Fig. 5 is the same as the layer 2 crossbar in Fig. 4, with the outputs generated using the comparators based neuron circuit in Fig. 1(a). The use of the two circuits in Fig 1 allows us to use the layer 2 memristor crossbar in both the forward and the backward pass.

*Step 3:* A training unit (see Fig. 6) takes the layer 2 errors, along with the layer 2 intermediate outputs ($DP_{L2,1}$ and $DP_{L2,2}$) to generate a set of training pulses. These pulses are applied to the layer 2 memristor crossbar to update the layer 2 synaptic weights.

*Step 4:* A process similar to step 3 is applied to update the synaptic weights in the layer 1 memristor crossbar. The layer 1 errors, along with the layer 1 intermediate outputs ($DP_{L1,1}$ and $DP_{L1,2}$, etc) are used as shown in Fig. 7.

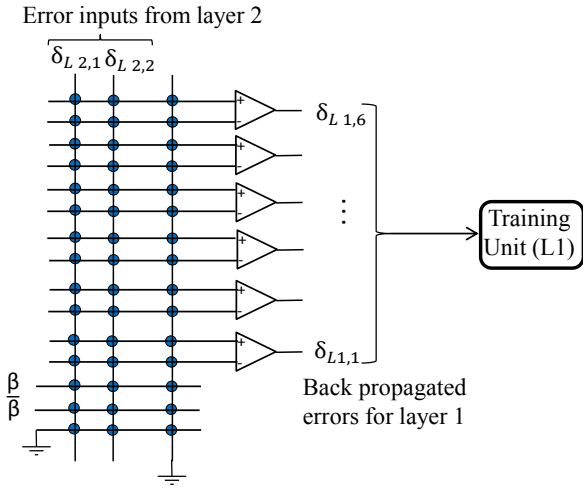



Fig. 5. Schematic of the neural network shown in Fig. 3 for back propagating errors to layer 1.

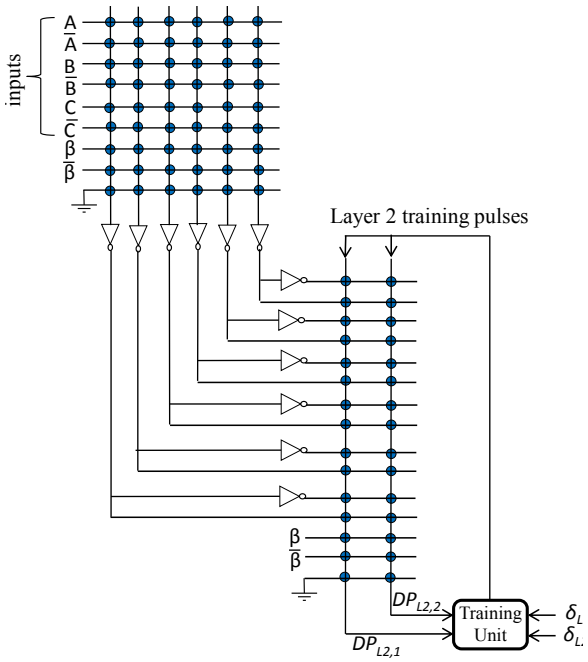



Fig. 6. Schematic of the neural network shown in Fig. 3 for updating layer 2 weights.

Fig. 8 shows the overall back-propagation training circuit, combining the circuits in Figs. 4 through 7. The training units in Figs. 6 and 7 have not been reproduced in Fig. 8. A set of pass transistors have been added to the second layer memristor crossbar that are controlled by signals *ctrl1* and *ctrl2*. *Ctrl1* enables forward propagation through the layer 2 memristor crossbar to generate neuron outputs, while *ctrl2* enables back-propagation to generate error values. Table I shows the state of these control signals for each of the four steps earlier. In step 2, *ctrl1* is set to 0 to isolate the second layer crossbar from the first layer crossbar to enable error generation.

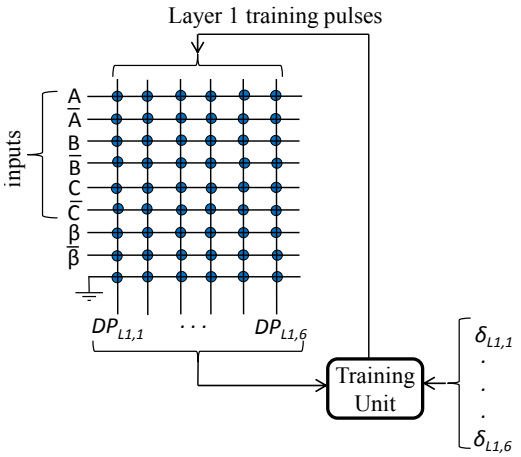



Fig. 7. Schematic of the neural network shown in Fig. 3 for updating layer 1 weights.

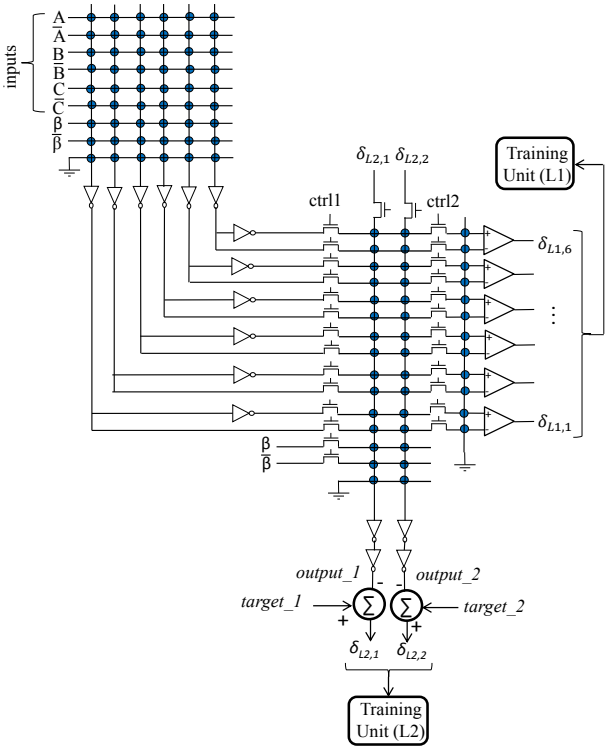



Fig. 8. Combined Schematic of the neural network shown in Fig. 3

TABLE I
CONTROL SIGNAL VALUES IN FIG. 9.

| | *ctrl1* | *ctrl2* |
|---|---|---|
| Step 1 | 1 | 0 |
| Step 2 | 0 | 1 |
| Step 3 | 1 | 0 |
| Step 4 | 0 | 0 |

## *E. Memristor weight update approach*

This section presents details on the memristor weight update technique utilized by the training unit. In the weight update equation (eqn. 5), the term $\eta$ is constant, while the terms $\delta$ and $x$ are discrete, with values of +1 or -1. The key impact of $\delta$ and $x$ are to determine the direction of the weight

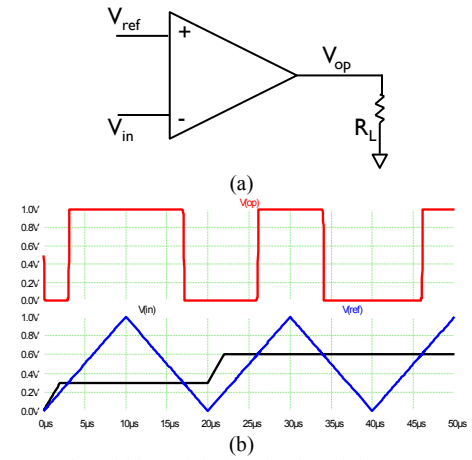



Fig. 9. (a) Pulse width modulation circuit and (b) waveform shows different pulse widths for different input magnitudes.

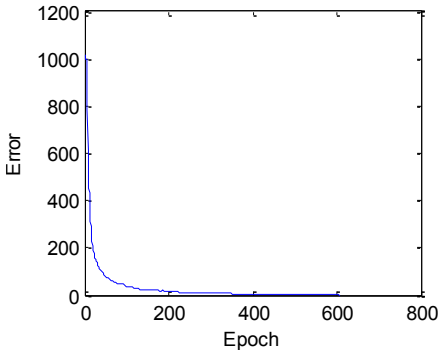



Fig. 10. Training curve approximating $1/(1+DP_j^2)$ as $g(DP_j)$ where $g(DP_j)=1-|DP_j|$ if $|DP_j|<0.95$ otherwise 0.05.

TABLE II
WEIGHT UPDATE DIRECTION.

| Input | Error | $\Delta w$ |
|---|---|---|
| +ve | +ve | Increase |
| +ve | -ve | Decrease |
| -ve | +ve | Decrease |
| -ve | -ve | Increase |

update. The magnitude of the weight update is determined by $DP_j$ and is the same for all the synapses of a particular neuron $j$.

All the weights within an entire crossbar can be updated in four steps. Each step considers one of the possible combinations of $\delta$ and $x$ and updates all memristors that are affected by that specific combination. The four possible combinations of $\delta$ and $x$ are shown in Table II along with the direction that the weight is changed.

If a voltage greater than the memristor write threshold is applied across a memristor, its conductance increases or decreases depending on the polarity of the device (determined by the fabrication process). To make desirable changes in memristor conductances, we need to apply a voltage of appropriate magnitude and polarity for a suitable duration across the memristor.

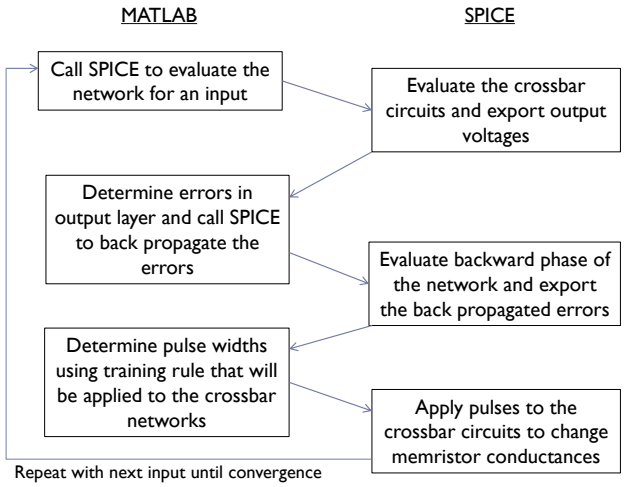



Fig. 11. Block diagram displaying how the MATLAB and SPICE simulation platform was used to train and evaluate the neuron circuit.

In this study, we have varied the pulse width to get different amounts of updates in the memristor conductances. One option for determining the training pulse width is to sense the dot product value ($DP_j$), discretize it and use it to determine the pulse width or number of pulses. Another option is to sense the analog dot product and use it directly to determine the pulse width through appropriate analog circuits. Fig. 9 shows a tentative analog circuit for pulse width modulation which could be used to pulse the memristor crossbars. Here a smaller $V_{in}$ generates a longer pulse and vice versa.

To simplify the training hardware, in the training rule (eqn. 4) we can approximate $1/(1+DP_j^2)$ as a piecewise linear function. Fig. 10 shows the training curve for the MNIST dataset (with 1000 inputs) when approximating $1/(1+DP_j^2)$ as $g(DP_j)$ where $g(DP_j)=1-|DP_j|$ if $|DP_j|<0.95$ (0.05 otherwise). For this experiment a two layer network having 20 hidden neurons and 10 output neurons was utilized. The error in Fig. 10 approaches zero, indicating that this approximation was valid.

## IV. Experimental Setup

We evaluated all operations on the memristor crossbars using SPICE circuit level simulations. As opposed to carrying out the simulation in high level tools, such as MATLAB, evaluating the actual memristor circuit in SPICE allows for more accurate modelling of the memristor grid. The alternate current paths and wire resistances within the crossbar are simulated. Our studies show that training a crossbar without the presence of alternate current paths leads to improper operation when the final weights are simply written to the memristors in the circuit. We utilized an accurate memristor device model [27] for the simulations.

MATLAB and SPICE were used to develop a simulation framework for applying the back-propagation learning algorithm to a multi-layered neuron circuit. Fig. 11 shows the simulation flow. The circuit was evaluated by applying an input pattern. For each input, the output and dot products ($DP_j$) were recorded from the SPICE simulation. Errors in the output layer were determined in MATLAB and the back-propagated errors were determined using the SPICE circuit. These voltages ($DP_j$ and errors) along with the neuron inputs are used to determine the amount of change that must be applied to each weight in MATLAB. To change the weights, a set of pluses is applied to the SPICE circuit based on pulse widths determined by the leaning algorithm. The circuit is evaluated again with the updated weights, and the cycle continues until the error is minimized.

It is assumed that this entire training process would be managed through hardware surrounding the memristor crossbars. The actual hardware design that will record the inverter inputs (or dot products) and generate pulses of appropriate width for updating the memristor conductances will be studied in more detail in future work.

TABLE III
Truth table of three input odd parity function and the encoding used for inputs and output for our implementations.

| Inputs | Output | Encoded inputs | Encoded output |
|---|---|---|---|
| 0 0 0 | 0 | -1 -1 -1 | -1 |
| 0 0 1 | 1 | -1 -1 1 | 1 |
| 0 1 0 | 1 | -1 1 -1 | 1 |
| 0 1 1 | 0 | -1 1 1 | -1 |
| 1 0 0 | 1 | 1 -1 -1 | 1 |
| 1 0 1 | 0 | 1 -1 1 | -1 |
| 1 1 0 | 0 | 1 1 -1 | -1 |
| 1 1 1 | 1 | 1 1 1 | 1 |

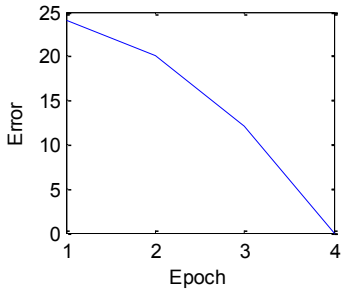



Fig. 12. Learning curve for three input odd parity function on a two layer network.

## V. Results

### A. Two Layer Network

**Three Input Odd Parity Function:** We have utilized a two layer network having six neurons in the hidden layer and one neuron in the output layer for training the non-linearly separable three input odd parity function. Table III shows the truth table of the three input odd parity function and the encodings used for inputs and output for our implementations. The schematic of the circuit is similar to that in Fig. 8, except that there is only one output instead of two. Fig. 12 shows the training curve obtained from MATLAB-SPICE simulation utilizing the back-propagation algorithm stated in section III. After four epochs, the recognition error becomes zero, indicating that the training was successful.

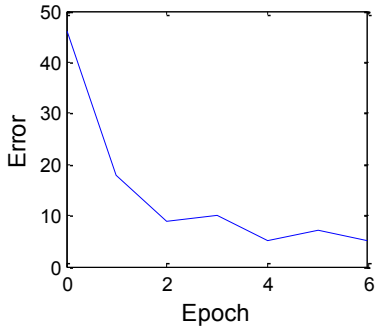



Fig. 13. Learning curve for training on Wisconsin breast cancer dataset.

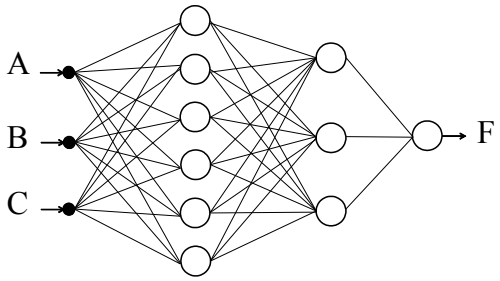



Fig. 14. Three layer network for learning three input odd parity function.

TABLE IV
RECOGNITION ERROR ON TEST DATA ON WISCONSIN BREAST CANCER DATASET.

| | Recognition error (%) |
|---|---|
| Benign | 8 |
| Malignant | 7 |

**Wisconsin Breast Cancer Dataset:** We have utilized the Wisconsin breast-cancer dataset [28] consisting of 699 patterns of two classes (benign and malignant). Each pattern consists of 9 attributes/features which were normalized such that the maximum attribute magnitude was 1. Although the inputs to the first layer were continuous, all the intermediate neuron outputs and errors were discretized as described earlier. We have trained a two layer neural network having 6 neurons in the hidden layer and one in the output layer. Utilizing the BP learning algorithm we have trained a memristor crossbar in our MATLAB-SPICE platform taking 200 patterns and were able to achieve less than 3% error. Fig. 13 shows the learning curve and Table IV shows the percentage of recognition error on another 200 test patterns. We can observe that on the test dataset, the recognition error is less than 8%.

### *B. Three Layer Network*

We utilized a three layer network (Fig. 14) for learning the three input odd parity function. This network consists of 6 neurons in the first hidden layer, 3 neurons in the second hidden layer, and one output neuron. Fig. 15 shows the circuit schematic for this network and Fig. 16 shows the training curve obtained from a MATLAB-SPICE simulation. The results indicate that the system learned the function with zero error.

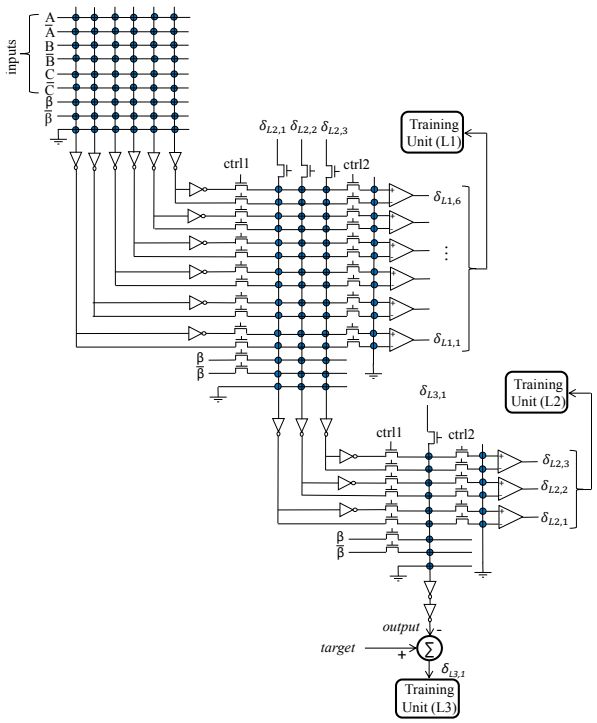



Fig. 15. Schematic of the neural network shown in Fig. 14.

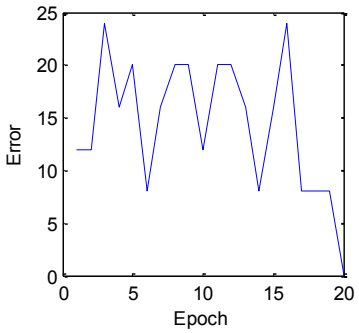



Fig. 16. Learning curve for three input odd parity function on three layer network.

## VI. CONCLUSION

This paper demonstrates that it is possible to implement back-propagation based training of memristor crossbar circuits directly into hardware. This implies that it is feasible to build memristor crossbar based neuromorphic processing tiles that are able to learn complex data sets directly through low level hardware circuits. Memristor crossbar based neuromophic processors have been shown to be 5 orders of magnitude more energy efficient than high performance Intel Xeon processors [1].

In this paper we have demonstrated training non-linearly separable functions using memristor crossbar circuits through back-propagation. We have examined 2 layer and 3 layer neural networks and shown training of non-linearly separable functions on these networks. As future work, we will investigate the implementation of neural networks with a larger number of layers and large numbers of neurons per layer using memristor crossbar circuits.

## REFERENCES


[1] T. M. Taha, R. Hasan, C. Yakopcic, and M. R. McLean, "Exploring the Design Space of Specialized Multicore Neural Processors," IEEE International Joint Conference on Neural Networks (IJCNN), August 2013.

[2] B. Belhadj, A. J. L. Zheng, R. Héliot, and O. Temam. "Continuous real-world inputs can open up alternative accelerator designs," SIGARCH Comput. Archit. News 41, 3 (June 2013)

[3] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in Proceeding of the 38th Annual International Symposium on Computer Architecture, 2011, pp. 365–376.

[4] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, O. Temam, "BenchNN: On the Broad Potential Application Scope of Hardware Neural Network Accelerators," IEEE International Symposium on Workload Characterization (IISWC), November 2012.

[5] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," Technology@Intel Magazine, Feb. 2005.

[6] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," International Symposium on Microarchitecture (MICRO), 2012.

[7] L. O. Chua, "Memristor—The Missing Circuit Element," IEEE Transactions on Circuit Theory, 18(5), 507–519 (1971).

[8] D. Chabi, W. Zhao, D. Querlioz, J.-O. Klein, "Robust Neural Logic Block (NLB) Based on Memristor Crossbar Array" IEEE/ACM International Symposium on Nanoscale Architectures, pp.137-143, 2011.

[9] C. Zamarreño-Ramos, L. A. Camuñas-Mesa, J. A. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," Frontiers in Neuroscience, Neuromorphic Engineering, vol. 5, pp. 1-22, Article 26, Mar. 2011.

[10] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing Memristor found," Nature, 453, 80–83 (2008).

[11] S. H. Jo, K.-H. Kim, and W. Lu, "High-Density Crossbar Arrays Based on a Si Memristive System" Nano Letters, 9(2), 2009, pp. 870-874.

[12] G. S. Snider, "Cortical Computing with Memristive Nanodevices," SciDAC Review, (2008).

[13] Russell, S. & Norvig, P. (2002). Artificial Intelligence: A Modern Approach (2nd Edition). Prentice Hall, ISBN-13: 978-01379039555.

[14] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," Journal of Neural Computation, Vol. 22, issue 12, 3207-3220, 2010.

[15] S. B. Furber, S. Temple and A. D. Brown, "High-Performance Computing for Systems of Spiking Neurons," Proceedings of AISB'06 workshop on GC5: Architecture of Brain and Mind, vol.2, pp 29-36, Bristol, April, 2006.

[16] J. Schemmel, J. Fieres, K. Meier, "Wafer-Scale Integration of Analog Neural Networks", IEEE International Joint Conference on Neural Networks (IJCNN), 2008.

[17] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," IEEE Custom Integrated Circuits Conference (CICC), vol., no., pp.1-4, 19-21 Sept. 2011.

[18] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, D. S. Modha, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," The International Joint Conference on Neural Networks (IJCNN), pp.1-8, June 2012.

[19] G. Indiveri, B. L. Barranco, T. J. Hamilton, A. van Schaik, R. E. Cummings, T. Delbruck, S. C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. S. Gotarredona, J. Wijekoon, Y. Wang, and k. Boahen, "Neuromorphic silicon neuron circuits," Frontier of Neuroscience, 2011.

[20] P. A. Merolla, K. Boahen, "Dynamic computation in a recurrent network of heterogeneous silicon neurons," Proceedings of IEEE International Symposium on Circuits and Systems, May 2006.

[21] J. H. B. Wijekoon , P. Dudek, "Compact silicon neuron circuit with spiking and bursting behaviour," Proceedings of Neural Networks, Volume 21, Issues 2–3, Pages 524-534, 2008.

[22] J. Lin, P. Merolla, J. Arthur, K. Boahen, "Programmable Connections in Neuromorphic Grids," IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), pp.80-84,Aug. 2006.

[23] C. Zamarreño-Ramos, L. A. Camuñas-Mesa, J. A. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," Frontiers in Neuroscience, Neuromorphic Engineering, vol. 5, pp. 1-22, Article 26, Mar. 2011.

[24] D. Chabi, W. Zhao, D. Querlioz, and J.-O. Klein, "Robust neural logic block (NLB) based on memristor crossbar array," in Proc. NANOARCH, 2011, pp.137-143.

[25] D. Chabi, W. Zhao, D. Querlioz, J-O. Klein, "Robust Neural Logic Block (NLB) based on Memristor Crossbar Array," IEEE/ACM ISNA, pp. 137-143 (2011).

[26] F. Alibart, E. Zamanidoost, and D.B. Strukov, "Pattern classification by memristive crossbar circuits with ex-situ and in-situ training", Nature Communications, 2013.

[27] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor SPICE Model and Crossbar Simulation Based on Devices with Nanosecond Switching Time," IEEE International Joint Conference on Neural Networks (IJCNN), August 2013.

[28] http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)