Hybrid Spiking-based Multi-layered Self-learning Neuromorphic System Based On Memristor Crossbar Arrays

Amr M. Hassan[1], Chaofei Yang[1], Chenchen Liu[1], Hai (Helen) Li[1], Yiran Chen[1]
[1]Electrical and Computer Engineering Department, University of Pittsburgh.
Emails: {amm418, chy61, chl192, hal66, yic52}@pitt.edu.

Abstract— Neuromorphic computing systems are under heavy investigation as a potential substitute for the traditional von Neumann systems in high-speed low-power applications. Recently, memristor crossbar arrays were utilized in realizing spiking-based neuromorphic system, where memristor conductance values correspond to synaptic weights. Most of these systems are composed of a single crossbar layer, in which system training is done off-chip, using computer based simulations, then the trained weights are pre-programmed to the memristor crossbar array. However, multi-layered, on-chip trained systems become crucial for handling massive amount of data and to overcome the resistance shift that occurs to memristors overtime. In this work, we propose a spiking-based multi-layered neuromorphic computing system capable of online training. The system performance is evaluated using three different datasets showing improved results versus previous work. In addition, studying the system accuracy versus memristor resistance shift shows promising results.

I. Introduction

With rapidly growing of online data, there is an urging need for new systems that can provide real-time data processing with high power efficiency. Traditional von Neumann systems are not able to cope up with such demands due to the well known "memory wall" phenomenon [1]. Neuromorphic computing systems, instead, offer great potentials by mimicking the working mechanism of human brains, where data are stored and processed at the same location [2]. Many research entities around the globe adopted those systems. For example, the IBM TrueNorth system [3] implement artificial synapses by using conventional CMOS circuitry [4, 5]. However, the high implementation cost could be a great challenge for further performance improvement and system scaling up.

Emerging memory devices [6], such as Resistive Random Access Memories (ReRAM), or Memristors, can offer a solution to this problem. Given their low feature size and synaptic like behavior [7, 8], memristors have been heavily investigated for possible usage in implementing large-scale neuromorphic systems. The two-terminal device is used to construct crossbar arrays, where memristor cells are located at each intersection of vertical and horizontal metal wires. Such a layout is of close resemblance to neural network models, in which the conductances of memristor cells correspond to the synaptic weights. In addition, the conductance value can be programmed to different discrete levels (or even continuous levels) by adjusting the amplitude (or pulse width) of the programming signal [9, 10].

Memristor crossbar arrays were used to implement matrix-vector multiplication in the neuromorphic computation process [11–13], in which voltage and currents values are used to represent the actual input and output data. These systems rely on Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs) as communication blocks. Consequently, further up scaling of these systems are hindered as the design complexity, cost, and area will rapidly increase. Alternatively, spiking-based neuromorphic systems were developed, where the inputs (outputs) take the form of voltage (current) spikes [14–16]. Such systems utilize simple CMOS circuitry to digitize the inputs and outputs to spikes, thus, offering adequate performance without the need for complex and expensive communication blocks.

The system in [15] is a single layer pattern recognition system based on what is so called off-chip (sometimes called offline or *ex-situ*) training. In this method, computer-based simulations are used to train the neural network and obtain the corresponding memristive weights. After that, those weights are programmed to the memristor cells in the crossbar array. Despite showing good compromise between performance and low hardware utilization, single-layered, off-chip trained systems cannot cope up with the overwhelming increase in the amount of data to be processed nowadays [12]. Moreover, the memristors resistance values drift from the programmed value during normal read operation, which requires regular weights updating for proper operation; a thing that off-chip training techniques cannot provide. That is why on-chip (sometimes called online or *in-situ*) training is widely adopted recently [12, 13], where extra hardware is added to the chip to allow online training (using training techniques which will be discussed later in Section III-*A*) and constantly allow memristive weights updating.

To overcome the previous problems, we propose a hybrid spiking-based multi-layered self-learning neuromorphic system. The proposed system performance were evaluated from different aspects and following are the main contributions:

- Introducing a simple and effective method of implementing backpropagation in hardware, which reduces the time needed for circuit training into half compared to previously published work.
- Offering design methodology and in-depth study for multi-layer spiking-based neuromorphic pattern recognition systems.
- Improving average failure error by 42% than previously published work for three different datasets.

To the authors best knowledge, such a hybrid system has never been proposed before.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

The rest of the paper will be organized as follows: Section II introduces the basic building blocks of the system and the implementation fundamentals. Section III explains the proposed multi-layer system, the design considerations, and the simple way of implementing back-propagation. Section IV evaluates and discusses the use of our design in three different test cases. At the end, Section V concludes the paper.

II. Preliminary

A. Memristor Devices and Used Model

Memristors are nonlinear devices whose resistance values change when the voltage across the device exceeds a certain threshold [10]. Usually they are made of an oxide layer sandwiched between two conductive layers, where TiO_{2-x} [17] and a layer composed of HfO_x and AlO_x [10] being the most promising oxides used. Since high energy efficient crossbar is needed, devices with higher resistances range become a must. The latter oxide shows a resistance ratio of ($R_{OFF}/R_{ON}\approx1000$) and on-resistance ($R_{ON}\approx10K\Omega$), which is adequate to be utilized in our system. For crossbar arrays, each cell is composed of a memristor and another device to enhance selectivity and eliminate *sneak path* problem [18, 19]. Among the different devices used as selectors [15, 20], the 1-Transistor 1-Resistor (1T1R) structure has been proven to be more reliable and has negligible effect on the overall conductance of the cell [15].

For this work, we are going to adopt the 1T1R structure. The memristor resistance ranges from [50KΩ, 1MΩ], which is divided into 8 discrete levels [15]. Those are the only allowable levels during training.

B. Neural Network Crossbar Arrays

Fig. 1 illustrates the resemblance between neural networks and crossbar arrays, where a typical one layer neural network can be implemented using memristor crossbar arrays and CMOS circuity. The synaptic weights, w_{ij}, which connect the inputs (blue circles) and output (gray circles) neurons in Fig. 1(a), are represented by the conductance, G, of the memristor cells at each cross point of the array in Fig. 1(b). The output neurons (and hidden neurons, if any) perform two functions (i) they evaluate the weighted sum of the inputs, $z_j = \sum_{i=1}^{N} x_i w_{ij}$, and (ii) they generate the output according to a nonlinear activation function, $y_j = f(z_j)$. The weighted summation function can be implemented directly using the crossbar array as follows: assuming that each vertical (bit) line is virtually connected to ground (which can be done using the Integrate-and-Fire Circuit (IFC) [15]), the inputs, x, are encoded into a series of voltage pulses, V, and applied to the horizontal (word) lines. Those pulses will be multiplied by the conductance, G, resulting in current $I = GV$ injected in each bit line. The second function can be done using the IFC, which will be explained in Section II-*C*.

Since the synaptic weights can be either positive or negative, two memristors will be used to represent one synapse, where $G_{ij} \equiv G_{ij}^{+} - G_{ij}^{-}$. In such a scheme, each output neuron is represented by two columns, representing G^+ and G^-, and currents

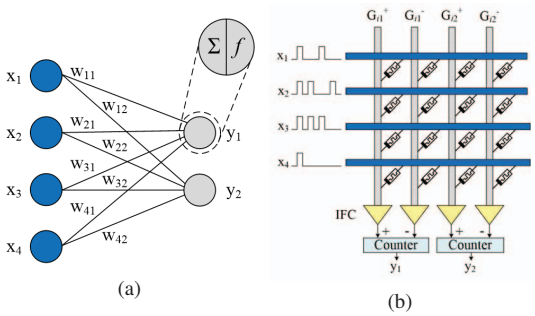


Fig. 1. Neural network implementation using memristor crossbar arrays. (a) shows a conventional diagram for 4×2 neural network while (b) shows the crossbar implementation of that network.

from those columns are directed to IFC, where the corresponding pulses are generated. Those pulses are then used to increment (in case of G^+) or decrement (in case of G^-) the value of a counter which represent the final output of the neuron. In addition, this exponentially expands the number of allowable resistance levels to 57 levels instead of 8 only, which are all the possible combinations 8^2 minus the number of repeated combinations of $G_{ij} = 0$ (7 combinations).

The major advantage of the spiking based systems is that it does not need expensive hardware, such as ADCs or summer amplifiers, unlike the other level based systems [12, 13].

C. Integrate-and-Fire Circuit (IFC)

Fig. 2(a) shows the schematic of the IFC as proposed in [15]. This circuit generates number of voltage spikes at the output, V_{out}, which is proportional to the magnitude of the input current, I_{in}, that comes from the crossbar column j.

We characterize the IFC in the curves shown in Fig. 2(b). The curves show nonlinear dependency on the input current, I_{in}, with the ability of controlling the maximum number of spikes, IFC_{max}, by adjusting the reference voltage, V_{ref} [15]. We will make use of these two properties to represent the activation function of hidden neurons and in designing the intermediate stages IFC, as will be further discussed in Section III-*B* and IV-*A*.

III. Design Methodology and Hardware Implementation

A. Proposed Training Scheme

Neural networks in general need to be trained to provide proper functionality. The most used training scheme for multilayer neural networks is the back-propagation algorithm [21]. In the recent period, many variants of that algorithm have been proposed and used in training memristor crossbar arrays [13, 22]. In this work, we will introduce, for the first time, a modified training algorithm for hybrid spiking-based multilayered systems. This algorithm has reduced the time needed for training by half compared to previously published work, which shall be explained in Section III-*B*.

In order to apply the back-propagation algorithm, input has to be fed forward through the whole network to calculate the

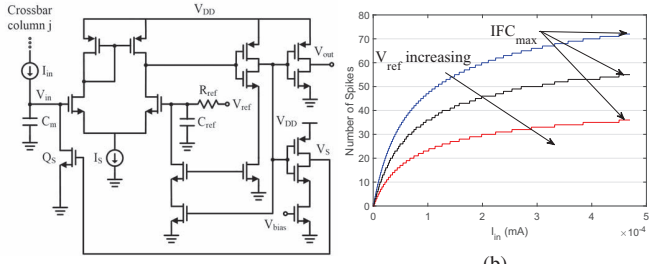


Fig. 2. Integrate-and-Fire Circuit. (a) shows the circuit diagram of the IFC as proposed in [15], while (b) shows our simulation for that circuit for the number of spikes (y-axis) against current values (x-axis) for different V_{ref}.

weighted sums and activations for all the hidden and output neurons. Once that is done, weight update is calculated according to the following formula [21]:

$$\nabla w_{ij} = \eta \times \delta_j \times x_i, \quad (1)$$

where η is the learning rate, x_i is the i^{th} output of the previous layer neuron (input or hidden), and δ_j is the j^{th} error propagated form the next layer (hidden or output) and is expressed as [21]:

$$\delta_j = \begin{cases} (t_j - y_j) & for\ the\ output\ layer \\ f'(z_j) \sum_{k=1}^{N} w_{kj}^n \delta_k^n & for\ any\ hidden\ layer, \end{cases} \quad (2)$$

where t_j (y_j) is the j^{th} desired (actual) output at the output layer, f' is the derivative of the activation function of the hidden layer, z_j is the j^{th} weighted sum, and the superscript n refers to the next layer. So, it can be seen that weights updates are obtained backwards starting from the output layer. Applying the exact version of back-propagation algorithm requires very expensive hardware, such as ADCs, DACs, lookup tables (to calculate the activation function derivative) , buffers, and multipliers.

In this work, we propose a simpler, yet accurate enough, version of back-propagation algorithm. For spiking systems, the inputs to any layer are always positive and in form of spikes. By using this property, we can, approximately, separate Eq. (1) to magnitude and sign, where the sign is determined by $sgn(\delta_j)$ and the magnitude by $\eta \times x_i$. In that way, detecting the sign information of δ_j doesn't require complex hardware, which significantly simplifies the design of the training circuits, as will be seen later. Algorithm 1 depicts the exact steps of the proposed training scheme.

B. Hardware Implementation of Two Layers Crossbar Array

In this part, we will discuss the actual implementation of the proposed algorithm in hardware. Shown in Fig. 3 is an example of 4×3×2 neural network (lower left corner) and its memristor crossbar array implementation. All the corresponding elements between the neural network diagram and the hardware implementation are color matched.

1) The feed forwarding step for error vector calculation. During the feed forward step of the proposed algorithm, all the switches with *ctrl 1* (*ctrl 2*) are closed (opened). A randomly

Algorithm 1 Proposed Training Algorithm

1: Randomly initialize memristor weights
2: **while** $(E > E_{desired})$ **do**
3: **for** (each x in the training set) **do**
4: Feed the input x forward through the whole network.
5: Calculate z_j and y_j for all hidden and output neurons.
6: Calculate the error for each output layer neuron j using: $\delta_j = sgn(t_j - y_j)$
7: Back propagate the above error for hidden layers neurons.
8: Calculate the error for each hidden layer neuron j by using: $\delta_j = sgn\left(f'(z_j) \sum_{k=1}^{N} w_{kj}^n \delta_k^n\right)$
9: Apply programming pulses to memristors, where the number of pulses for the i^{th} and j^{th} memristor in each crossbar are determined by $\eta \times x_i$, and the polarity by δ_j.
10: **end for**
11: **end while**

chosen input x from the training dataset is then converted into pulses and applied to the word lines of the first crossbar layer. Positive and negative currents are then converted by IFC into spikes and then applied to the up and down control signals of the counter, respectively. We are making use of the nonlinearity in the IFC characteristics and counter, whose lower limit value is zero, to represent the nonlinear activation function of the hidden neuron. The value stored in the counters are then converted into pulses and applied to the word lines of the second crossbar layer as well. After evaluating the output vector y, the desired output vector t is compared with the y and the sign of the error vector δ^{L2} is obtained. The superscript in the error vector refers to the crossbar layer number.

2) The backpropagation step for crossbar weight updating. According to (2), for any hidden layer, the summation part can be thought of as the error vector δ^{L2} multiplied by the transpose of the conductance matrix of the second crossbar layer G_2'. So δ^{L2} will be applied vertically, from the top, on the bit lines and currents will be collected horizontally, on the left, from word lines. Since the crossbar array is originally designed with two memristors per synaptic weight and for feed forward propagation, δ^{L2} has to be modified before applying to the crossbar. The error mapping block generate two error

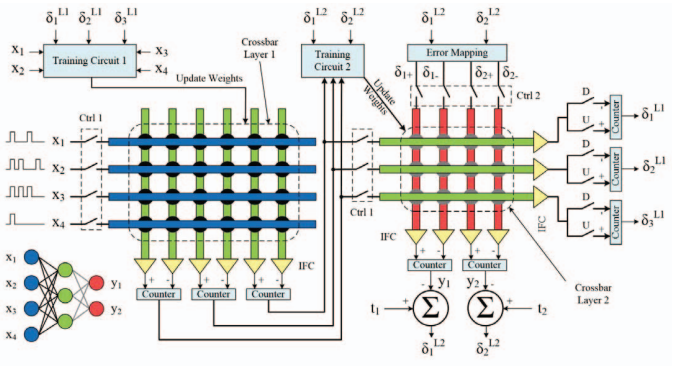


Fig. 3. Hardware implementation of a two layer memristor crossbar array. A 4×3×2 neural network diagram is shown in the lower left corner, and its circuit implementation is shown in the rest of the figure.

vectors, which will be applied sequentially to second crossbar. Each one of those vectors is double the size of the original error vector and mapping is done according to Table I. Each δ_j can take only three values: +1, 0, -1. Two vectors, and two corresponding values per vector δ_{j+} and δ_{j-}, are generated for each value δ_j. During the application of the two generated error vectors, the switches with *ctrl 2* (*ctrl 1*) are closed (opened). The top row corresponding to each of the three values in Table I is applied first and the switched marked as U (D) are closed (opened), while the bottom row is applied second and the switches marked as D (U) are closed (opened). After that, the sign of the error vector δ^{L1} is obtained, which is merely the MSB of the counter (for polarity) and a zero detector.

Each crossbar has its own weight updating block (training circuits in Fig. 3). Each training block takes the corresponding input vector x and the error vector δ, then, weight updating is done on two steps, as shown in Fig. 4. Assuming the error vector, δ, and the amount by which each weight is updated, $\eta \times x$, as shown in the Fig. 4(b), the effective conductance G_{i1} (G_{i2}) needs to be increased (decreased). Since $G_{ij} = G_{ij}^{+} - G_{ij}^{-}$, the weight update is done deferentially with more priority for the conductance to be increased. For example, $\eta \times x_1 = 3$ which means both G_{11}^{+} and G_{12}^{-} (G_{11}^{-} and G_{12}^{+}) will increase (decrease) by 2 (1) levels. If any value in the error vector (input vector) is zero, then the memristors in the corresponding columns (row) are not updated. Fig. 4(b) shows the required memristors to be increased or decreased. Training is done in two steps: (i) weights to be increased, in which positive programming pulses are applied to the crossbar and the corresponding columns are connected to ground (Fig. 4(c)), and (ii) weights to be decreased, in which negative programming pulses are applied to the crossbar and the corresponding columns are connected to ground (Fig. 4(d)).

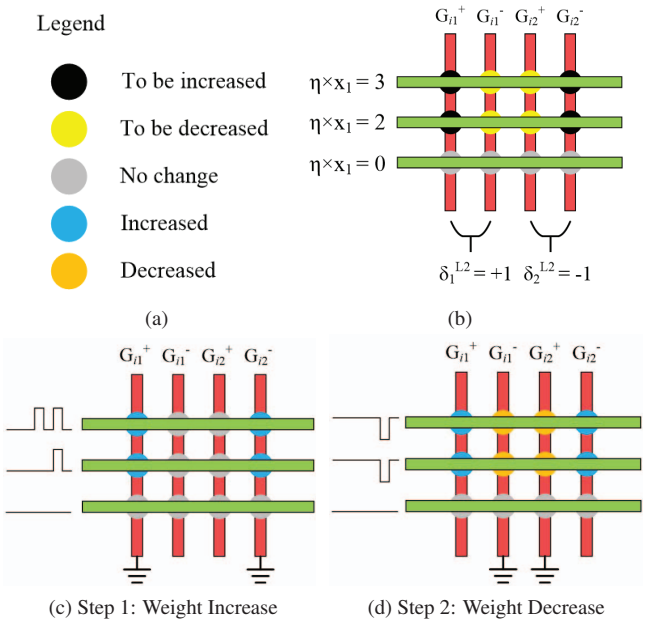


Fig. 4. Training the second crossbar in Fig. 1. (a) shows the legend for different colors, while (b) depicts the assumed value for the error vector δ and the input vector $\eta \times x$. (c) and (d) show the only two required steps to program the whole crossbar.

TABLE I
ERROR VECTOR MAPPING FOR DIFFERENT VALUES OF δ_j^{L2}

δ_j^{L2}	δ_{j+}^{L2}	δ_{j-}^{L2}	δ_j^{L2}	δ_{j+}^{L2}	δ_{j-}^{L2}	δ_j^{L2}	δ_{j+}^{L2}	δ_{j-}^{L2}
+1	1	0	0	0	0	-1	0	1
	0	1		0	0		1	0

To the authors best knowledge, no one has proposed a fast training scheme like this before. The only ones which are near to the proposed training scheme requires 4 steps to complete training [13, 22]. Since the activation function used in [13, 22] is a *tanh* function, the input x of the hidden neurons could be negative or positive; thus, increasing the different combination required for training into 4 instead of 2 as proposed here.

IV. System Evaluation

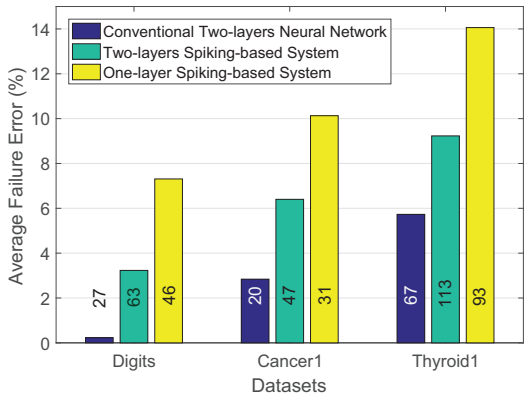
Three different datasets were used to evaluate the proposed training scheme. The first *Digits* dataset is simple binary figures corresponding to digits 0-5 from [15]. Training and testing ensembles were generated from the basic figures by randomly injecting errors in them [15]. The other two datasets are Cancer1 and Thyroid1 from the Proben1 database [23]. All the datasets were implemented using a two-layers spiking-based system and weights were updated according to the proposed algorithm in Section III. Table II summarizes the properties of the three datasets and the corresponding crossbar network configurations. For the *Digits* and *Cancer1* datasets, inputs are binary and 10 discrete levels, respectively, while inputs are analog for the *Thyroid1* dataset. In order to represent the inputs by spikes, each input of the Thyroid1 dataset is mapped between 0 ~ 1, then discretized into 10 levels, where each level corresponds to one spike.

In the following subsections, we will evaluate how much improvement did each of the three part of the proposed systems achieve, namely, extending the system to two-layers, controlling IFC_{max}, and finally online training.

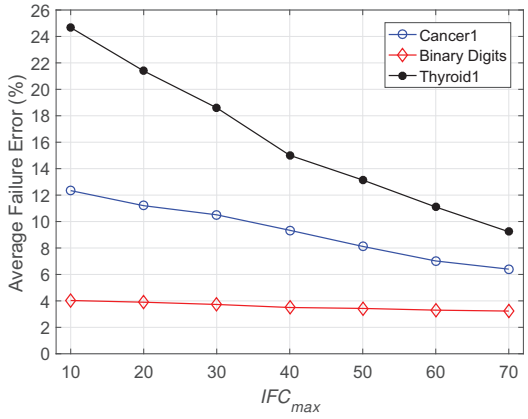
A. Ideal System Performance

Fig. 5(a) shows the performance of the three networks with respect to average failure error. For each system, a comparison is made among that proposed system, a one-layer spiking-based system, and a conventional two-layers neural network system. All simulations were done using MATLAB with the overall failure error averaged over 100 runs. As you can see, the two-layers system has adequately improved the overall failure error by 42% (on average) than the one-layer system and still within a reasonable fall behind range of the conventional neural network system.

The average number of epochs, averaged over 100 runs, to achieve this accuracy is shown inside the bar for each system. Each epoch represents one iteration of the while loop in 1. As shown, the epochs needed by the two-layers system to converge are more than those needed by the one-layer system. This is something expected since we are updating two crossbars instead of one.



(a)



(b)

Fig. 5. Ideal system evaluation for the architecture shown in Fig. 3. (a) shows a comparison between average failure error for conventional two-layers neural network (blue), two-layers spiking-based system (green), and one-layer spiking-based system (yellow). Results are shown for three datasets, namely Digits [15], Cancer1, and Thyroid1 [23], and (b) shows the average failure error for Digits (red), Cancer1 (blue), and Thyroid1 (black) versus IFC_{max} of the intermediate stages IFC.

As aforementioned in Section II-*C* that controlling IFC_{max} is very crucial for intermediate stages IFC, we further investigate the effect of IFC_{max} on the system performance. The two-layers spiking-based systems were evaluated for different value of IFC_{max}, as shown in Fig. 5(b). All simulations follow the same rules depicted before. It can be easily deduced that the performance of *Digits* system doesn't really change for different IFC_{max}. On the other hand, the performance of Cancer1 and *Thyroid1* systems dramatically depend on the IFC_{max} value. Such a behavior can be explained as follows: for the *Digits* system, the input resolution is binary, so it does not need intermediate stages IFC with high resolution. On the contrary, Both *Cancer1* and *Thyroid1* have higher input resolution (10 levels per input) which definitely needs high resolution for the intermediate stages IFC.

We couldn't increase IFC_{max} beyond 70 as the simulation time increases dramatically. It is worth mentioning that increasing IFC_{max} for intermediate stages IFC requires bigger counters, so the overall system area and power increase, especially for bigger crossbar arrays. Moreover, the hardware implementation of the system gets slower, as more spikes per input have to be applied sequentially for the second crossbar.

TABLE II
PROPERTIES OF THE USED DATASETS AND THEIR CORRESPONDING CROSSBAR SYSTEMS.

Dataset	Network Size	Training Set Size	Testing Set Size
Digits [15]	32×10×6	600	200
Cancer1 [23]	9×10×2	594	105
Thyroid1 [23]	21×15×3	6120	1080

B. Memristor Process Variations

For better and more realistic system evaluation, we are going to take into consideration memristor process variations. It is widely known that there is a deviation between the programmed and the desired memristor resistance due to process variations, in which most of those variations follow Gaussian distribution [24]. Those variations can dramatically degrade the system performance, especially for offline trained systems. Online training helps to significantly decrease the effect of process variations by constantly updating memristor weights on the fly. For our evaluation, we will assume that the memristor resistance is:

$$R_p = R_{desired} + \nabla R, \tag{3}$$

where R_p is the final programmed resistance, R_d is the desired resistance value, and $\nabla R \sim N(0, \sigma)$ is a random variable that follows a Gaussian distribution of mean value 0 and standard deviation σ. The simulation results for the two-layered spike-based systems using online (solid lines) and offline (dashed lines), are shown in Fig. 6(a). The average failure error, averaged over 100 runs, is plotted against the standard deviation σ, whose maximum value is restricted to 5% which is widely accepted [24]. For offline training, MATLAB was used to train each memristor resistance value only from the pool of allowable resistance states. Process variation modeling is applied when the trained resistance values are programmed to the memristor. On the other hand, online training takes into consideration process variation during each epoch of training. As shown in Fig. 6(a), online training has significantly improved the average failure error, where improvements at the worst process variation (σ = 5) are 21.97%, 17.95%, 16.18% for Thyroid1, Cancer1, and Binary Digits datasets, respectively. As the figure depicts, the rate of improvements increases as the process variations gets worse and is note the same for the three datasets. The latter can be explained as the sensitivity for process variations increase as the network size and input feature details increase.

The number of epochs, averaged over 100 runs, needed for training the online system versus σ is shown in Fig. 6. It can be concluded from this figure that the rate of increase of the number of epochs is not constant, and changes, approximately, exponentially according to the complexity of the used dataset. This explains why the number of epochs needed for *Cancer1* for σ = 1 and 2 is lower than the rest of the values.

V. CONCLUSIONS

In this work, we demonstrated a hybrid spiking-based multi-layered self-learning neuromorphic computing system. We

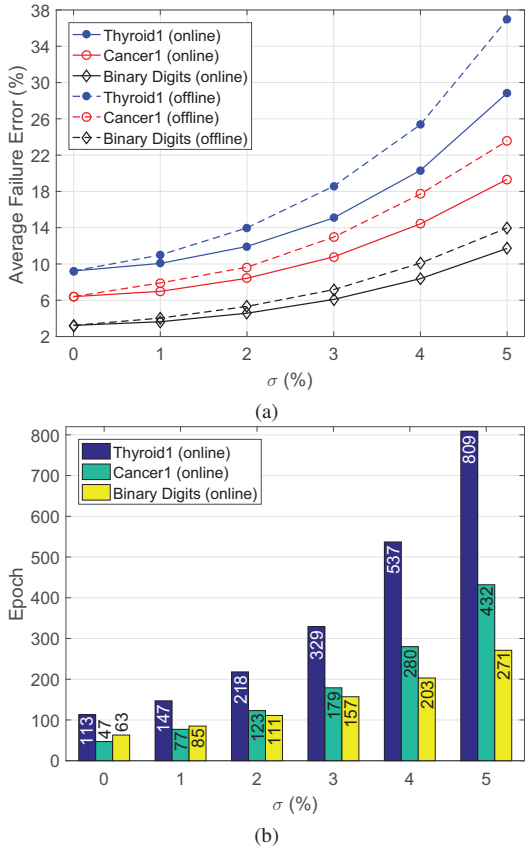


Fig. 6. System evaluation, after taking into account process variations, for the architecture shown in Fig. 3. (a) shows a comparison between average failure error for the two-layers spiking-based system, using online (solid) and offline (dashed) training, versus standard deviation σ for Thyroid1 (blue), Cancer1 (red), and Binary Digits (black). (b) shows the average number of epochs for online trained systems versus standard deviation σ for the same datasets.

also proposed a simple and effective online training algorithm for spiking systems, which reduces the required steps for weights updating into half of the previously published training algorithms. System evaluation, using three different datasets, showed overall performance improvement of 42% was achieved. Moreover, the online training algorithm shows adequate immunity against memristor process variations. Since the results shown here are promising, potential future work will be investigating the effect of increasing the crossbar size and the number of layers on the overall system performance.

VI. Acknowledgments

This work is in part supported by NSF CNS-1253424, XPS-1337198, and AFRL FA8750-15-2-0048.

References

[1] W. A. Wulf and S. A. McKee, "Hitting The Memory Wall: Implications of The Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, pp. 20–24, 1995.

[2] C. Gamrat, "Challenges and Perspectives of Computer Architecture at the Nano Scale," in *IEEE Comput. Soc. Annu. Symp. VLSI*, 2010, pp. 8–10.

[3] F. Akopyan *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, pp. 1537–1557, 2015.

[4] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling," in *IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2010, pp. 1947–1950.

[5] S. Mitra, S. Fusi, and G. Indiveri, "Real-Time Classification of Complex Patterns Using Spike-Based Learning in Neuromorphic VLSI," *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, pp. 32–42, 2009.

[6] International Technology Roadmap of Semiconductors, 2013 Edition, Emerging Research Devices. [Online]. Available: http://www.itrs.net/

[7] A. Thomas, "Memristor-Based Neural Networks," *J. Physics D Appl. Physics*, vol. 46, p. 093001, 2013.

[8] 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, pp. 1297–1301, 2010.

[9] T.-W. Lee and J. H. Nickel, "Memristor Resistance Modulation for Analog Applications," *IEEE Electron Device Lett.*, vol. 33, pp. 1456–1458, 2012.

[10] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating The Switching Dynamics and Multilevel Capability of Bipolar Metal Oxide Resistive Switching Memory," *Appl. Physics Lett.*, vol. 98, p. 103514, 2011.

[11] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor Crossbar-Based Neuromorphic Computing System: A Case Study," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, pp. 1864–1878, 2014.

[12] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, "Training Itself: Mixed-Signal Training Acceleration for Memristor-Based Neural Network," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 361–366.

[13] R. Hasan and T. M. Taha, "Enabling Back Propagation Training of Memristor Crossbar Neuromorphic Processors," in *Int. Joint Conf. Neural Networks (IJCNN)*, 2014, pp. 21–28.

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proc. IEEE*, vol. 102, pp. 652–665, 2014.

[15] C. Liu *et al.*, "A Spiking Neuromorphic Design with Resistive Crossbar," in *Design Automation Conference (DAC)*, 2015, pp. 14:1–14:6.

[16] M. Chu *et al.*, "Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron," *IEEE Trans. Ind. Electron.*, vol. 62, pp. 2410–2419, 2015.

[17] A. M. Hassan, H. A. Fahmy, and N. H. Rafat, "Enhanced Model of Conductive Filament-Based Memristor via Including Trapezoidal Electron Tunneling Barrier Effect," *IEEE Trans. Nanotechnol.*, vol. 15, pp. 484–491, 2016.

[18] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. Fahmy, and K. N. Salama, "Memristor Multiport Readout: A Closed-Form Solution for Sneak Paths," *IEEE Trans. Nanotechnol.*, vol. 13, pp. 274–282, 2014.

[19] C. Liu and H. Li, "A Weighted Sensing Scheme for ReRAM-Based Cross-Point Memory Array," in *IEEE Comput. Soc. Annu. Symp. VLSI*, 2014, pp. 65–70.

[20] B. Yan, A. M. Mahmoud, J. J. Yang, Q. Wu, Y. Chen, and H. H. Li, "A neuromorphic ASIC Design Using One-Selector-One-Memristor Crossbar," in *IEEE Int. Symp. Circuits and Syst. (ISCAS)*, 2016, pp. 1390–1393.

[21] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, New York, 2001.

[22] E. Zamanidoost, F. M. Bayat, D. Strukov, and I. Kataeva, "Manhattan Rule Training for Memristive Crossbar Circuit Pattern Classifiers," in *IEEE Int. Symp. Intell. Signal Process. (WISP)*, 2015, pp. 1–6.

[23] L. Prechelt, *"PROBEN 1: A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms"*. Univ., Fak. für Informatik, 1994.

[24] B. Liu *et al.*, "Digital-Assisted Noise-Eliminating Training for Memristor Crossbar-Based Analog Neuromorphic Computing Engine," in *Design Automation Conference (DAC)*, 2013, pp. 7:1–7:6.