Robust Neuromorphic Computing in the Presence of Process Variation

Ali BanaGozar[1], Mohammad Ali Maleki[1], Mehdi Kamal[1], Ali Afzali-Kusha[1], Massoud Pedram[2]
[1]School of Electrical and computer Engineering, University of Tehran
[2]Department of Electrical Engineering, University of Southern California
{ali.banagozar, m.a.maleki, mehdikamal, afzali}@ut.ac.ir, pedram@usc.edu

***Abstract*—In this paper, an approach for increasing the sustainability of inverter-based memristive neuromorphic circuits in the presence of process variation is presented. The approach works based on extracting the impact of process variations on the neurons characteristics during the test phase through a proposed algorithm. In this method, first, some combinations of inputs and weights (based on the neuromorphic circuit structure) are injected into the circuit and the features of the neurons are determined. Next, these features which are back-annotated, are utilized in an efficient ex-situ training approach to determine the proper weights of the neurons. The approach provides a considerable improvement in the output accuracy. To evaluate the effectiveness of the proposed approach, some approximate applications are studied using 90nm CMOS technology. The results of the study reveal that using this framework provides, on average, 17X higher output accuracy compared to the cases that the impact of the process variation is not considered at all.**

Keywords—Process Variation, Neuromorphic Computing, Training, Testing.

I. Introduction

Huge streams of data, which are nowadays produced on a regular basis, have to be processed so that they become intelligible and usable [1]. To accomplish the aforementioned goal, the data should be analyzed with sophisticated algorithms such as recognition, clustering, classification, and approximation. Processing this enormous size of information raises the need for high speed and energy efficient processing platforms. Accordingly, the issue of power consumption and proficiency should be addressed carefully.

The brain is the most power efficient, robust, swift, and highly parallelized organ in the body. Hence, plenty of endeavors at various levels of abstractions have focused on discovering capabilities of the human brain. Consequently, "Neuromorphic Computing" – a brand new (approximate computing) architecture – has emerged. The architecture involves two sub-disciplines. First one, known as "Machine Learning", is based on Classical Neural Networks (Classical NNs) and proposes an analytical model to emulate the functionality of the the brain. The second one tries not only to simulate brain's functionality, but to mimic the brain, and its biological neurons as well [2]. In the latter approach, information is coded as a combination of spikes. Thus, the resulting designs such as Spiking Neural Networks (SNNs) are highly energy efficient. Additionally, they provide more promising learning algorithms in the sense of continuous training, while the system is under execution (on-line learning). On the other hand, classical NNs are more mature and offer higher accuracy. Furthermore, if area cost constraints are imposed, classical NNs yield more area/energy efficiency compared to the SNNs [2]. Analog implementation of the classical NNs is becoming popular due to its improved speed and energy efficiency when it is exploited as an accelerator as a companion to conventional processors [3].

Synapses and neurons are two main constituents of neural networks. Synapses have weights whereas neurons have activation functions [4]. Low power consumption, resistive plasticity, as well as dense fabrication make the memristor a promising element for use in the analog implementation of the synapse [5]. Synapses could be implemented in various ways, among them, the memristor crossbar is a widely accepted solution for realizing synaptic arrays. There are various options to implement a neuron, three of which are analog comparator [6], operational amplifier (op-amp) [7], and inverter [8]. Among these implementations, the one based on inverter not only can operate at a lower energy level, compared to comparator or op-amp, but it also could be fabricated in a denser fashion.

Despite the fact that physical scaling according to Moore's law has resulted in many advantages (*e.g.*, lower operating voltage, higher switching speed, and smaller area/power consumption) in sub-100nm technologies, it has also elevated the significance of manufacturing process imperfections in the form of "process variations." In "sur-100nm" technologies, employing nominal transistor parameters may be considered as a valid assumption in designing a VLSI circuit. However, significant variations in transistor parameters, which have ensued from both manufacturing imprecisions and the complicated fabrication processes, invalidate the aforementioned assumption. Disparity in transistor's physical and electrical parameters translates into uncertainties in delay and energy consumption of logic gates and VLSI circuits [9]. Therefore, like any other circuit implemented in sub-100nm technologies, neuromorphic circuits are vulnerable to process variations as well. In this case, if the weights of the synapses are determined before manufacturing the chip (*i.e.*, ex-situ training) and based on the nominal transistors parameters, the output accuracy of the neuromorphic chip may be adversely affected. Note that one may employ in-situ training approach and extract the weights of the neurons after the chip is manufactured [6]. However, this approach is considerably more time consuming and effort intensive due to many access needed until the learning algorithm could converge to a (near-)optimum solution. Accessing the manufactured circuit during test phase has been reported as a solution to find the uncertainty of circuit components due to the process variation. In [10], an approach for extracting the impact of the process variations on the memristor crossbar of the neuromorphic chip has been presented. This approach is applied during the test phase, and

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

based on the collected information synaptic weights are mapped to proper places in a memristor crossbar.

In this work, we propose an approach to rapidly adjust the weights of the manufactured inverter-based memristive neuromorphic chips during the test phase for improving output accuracy. In this approach, first, the impact of process variations on the characteristics of the neurons is determined by applying some inputs and weights to the neuromorphic circuit while considering the circuit structure for each chip. Next, by applying a fast ex-situ training, based on the extracted information from the neurons, proper weights are determined. Finally, these weights are deployed on the chip. In order to assess the efficacy of the proposed framework, its impact on improving the accuracy of some approximate applications in the presence of process variation is studied.

The rest of the paper is organized as follows. In Section II, the impact of the process variation on synapses and neurons comprehensively is studied. Our proposed approach to improve the output quality of the neuromorphic chips in the presence of process variation is presented in Section III. In Section IV, the efficacy of the proposed approach is evaluated under different approximate applications. Finally, the paper is concluded in Section V.

II. Impact of Process Variation

In neural networks, neuron type (*i.e.*, activation function) and network topology (*i.e.*, number of layers and number of neurons in each one) are two systematic parameters which are determined based on the application type, distribution of inputs, and choice of the learning algorithm [11]. After deciding about these two parameters, the neural network is ready to be trained by an adequate amount size of training dataset until an acceptable output error rate is achieved (assuming that the training process converges). In the learning process, weights are tuned in such a way that the overall error of the network in response to the training dataset becomes as low as possible. After manufacturing the chip and during runtime, when the new unseen data is fed to the network, the output is guaranteed to be at its lowest possible cost (minimum error) when characteristics of the network components (*i.e.*, weights and activation functions) remain exactly the same as the ones derived in training phase. When the determined weights from ex-situ training (off the chip) are applied to the manufactured neuromorphic chip, any deviation in the behavior of circuit components (*e.g.*, inverter and memristor physical characteristics) may lead to sizeable accuracy loss. From a system level perspective, inverter represents activation function and memristors represent synaptic weights. Hence, in the rest of this section, we conduct two system-level studies to find the impact of the process variations on the output accuracy of the inverter-based memristive neuromorphic circuits. Fig. 1 shows the internal structure of this type of neuromorphic circuit. For implementing negative weights, the positive and negative of each input (differential input) is employed. Hence, a neuron in this structure consists of a chain of two inverters where the output of the first (second) inverter is considered as the positive (negative) input of the next layer. Note that neurons in the output layer only consist of one inverter.

The impact of the process variation on memristors [12] leads to weight fluctuations in networks. Thus, to show the effect of this uncertainty on the accuracy of the neuromorphic circuit, we add random noises to all the weights of the neurons of network and measure the network error in the presence of these changes. The noisy weight is defined by

$$Weight_{Noisy} = Weight_{Nominal} \times (1 + \sigma \times X) \quad (1)$$

where X is a normally distributed random number ($X \sim N[0,1]$), and σ is the standard deviation of the considered noise. This study has been performed on a system-level model of the circuits for a number of benchmarks. For this study, considered weight variations (*i.e.*, σ) are 5%, 15% and 25%, and in each weight variation, 1500 samples of the network are trained.

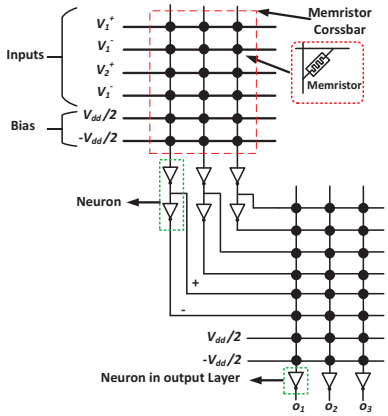

Fig. 1. A 2-layer inverter-based memristive neuromorphic circuit with two inputs.

When the inverter is employed as the neuron, the activation function of the neuron is modeled by hyperbolic tangent function defined by

$$f(x) = a + b \times \tanh((x - c) \times d) \quad (2)$$

where a, b, c and d are constant values that should be determined based on the Voltage Transfer Characteristic (VTC) of the corresponding inverter. Therefore, to extract the impact of the process variation on the neurons, first, we modeled the impact of the process variation on the VTC of the inverter in 90nm technology by using Monte-Carlo HSPICE simulation. Accordingly, a library containing 1500 VTC samples and their corresponding fitted hyperbolic tangent functions have been generated. Then, 1500 system-level samples of the network have been modeled where the activation functions of neurons have been selected randomly from the generated library. Eventually, each network was trained. Note that for this study, we assumed 5% physical parameter variation (*i.e.*,σ/μ) for the transistors of each inverter.

The Mean Square Error (MSE) of the outputs of the networks in these two studies are illustrated in Fig. 2. As the results show, the variation of the activation function has a significantly larger impact on the accuracy of the network outputs compared to that of weights. Note that to mitigate the impact of process variation on a memristive crossbar, one may utilize the techniques proposed by [12]–[14]. In this work, we focus on considering the impact of the process variations on the physical parameters of neurons.

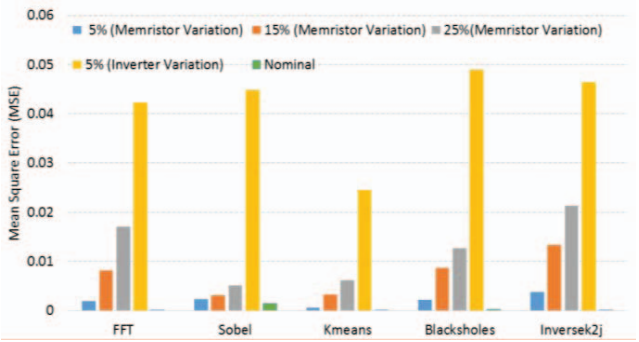

Fig. 2. MSE of inverter-based memristive neuromorphic circuits in the presence of process variation.

III. Proposed Framework

The learning problem for the neural networks is NP-complete [15], and algorithms such as backpropagation, can only determine local optima. Additionally, due to the process variations, activation functions of neurons are not deterministic. As a result, the activation functions should be statistically emulated in learning algorithm, which makes the learning problem more complex. Therefore, determining a set of weights that would lead to minimum output error for all manufactured neural network chips is a formidable task. Characterizing activation functions after chip manufacturing may lead to improving the quality of training, and hence, we propose a two-step approach. In the first step neuron characteristics (*i.e.*, VTC) from the manufactured chip are extracted, and in the second step, synaptic weights of the neuromorphic computing chip are adjusted based on the extracted information. In the following sub-sections, the details of these two steps are provided.

A. Extracting neuron characteristics

In order to extract the VTC of the neurons, we propose an algorithm, which its pseudo-code is presented in Fig. 3. In this algorithm by applying a variety of weights and inputs, and observing NN outputs, the VTCs of neurons are determined. The algorithm starts from neurons in the output layer, due to the observability of their outputs. The output of the j^{th} neuron in i^{th} layer (a_j^i) is defined by

$$a_j^i = \sigma\left(\sum_k w_{jk}^i a_k^{i-1} + \beta^i b^i\right) \quad (3)$$

where σ is the activation function, w_{jk}^i is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer to the j^{th} neuron in the i^{th} layer. Also, b^i is the bias in the i^{th} layer and β^i is its corresponding weight. In this work, $V_{DD}/2$ is considered for the bias (b^i) in all layers. Since, b^i is a predefined value and nearly constant, to capture the activation function of each output neuron, we suggest to set the all incoming weights to the last layer (w^L) to zero. Hence, the output of the neuron (in the output layer) becomes $\sigma(\beta^L b^L)$. Now, by applying some values to bias-connected weight and extracting tuples of (a^L, β^L), the neuron activation functions in the output layer can be determined by curve fitting (see Fig. 4(a)). Note that as the output of all neurons in the output layer are observable, we can extract the VTCs of all the neurons in this layer simultaneously.

As mentioned earlier, the activation function in the inverter-based neural network is modeled by a hyperbolic tangent function where the VTC of the positive and negative output of an inverter-based neuron in the considered 90nm technology is depicted in Fig. 5.

```
Set to '0' all the input weights of the output layer
Sweep β^L : From β_max,p to β_min,p by resolution α_p
    Observe outputs
End Sweep
Extract the activation functions
For each hidden layer, except the first hidden layer
    Do
        Select ξ_i neurons
        Set the proper weights to generate paths from the selected neurons to the output layer
        Sweep β^L : From β_max,p to β_min,p by resolution α_p
            Observe outputs
        End Sweep
        Extract the activation functions
        Set the proper weights to generate paths from the selected neurons to the output layer
        Sweep β^L : From β_max,n to β_min,n by resolution α_n
            Observe outputs
        End Sweep
        Extract the activation functions
    Till all the neurons in the hidden layer are observed
End For
Do
    Select ξ_i neurons from first layer
    Set the proper weights to generate paths from the selected neurons to the output layer
    Sweep a^l : From β_max,p to β_min,p by resolution α_p
        Observe outputs
    End Sweep
    Extract the activation functions
    Set the proper weights to generate paths from the selected neurons to the output layer
    Sweep a^l : From β_max,n to β_min,n by resolution α_n
        Observe outputs
    End Sweep
    Extract the activation functions
Till all the neurons in the first hidden layer are observed
```

Fig. 3. Pseudo code of the proposed VTC extraction algorithm

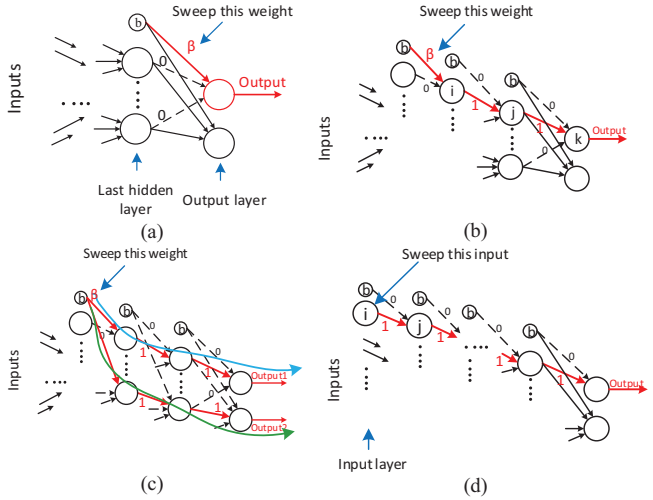

Fig. 4. Examples of the processing the VTC extractions of (a) neurons in output layer, (b) a neuron in hidden layer, (c) two neurons in hidden layer, and (d) a neuron in first hidden layer.

To determine the VTC curve and specifically the shape of inverters state change more accurately, we suggest to sweep β

around the state change. Due to the process variations, the steepness of the state change is altered. Thus, the range for sweeping β (*e.g.*, $[\beta_{min}, \beta_{max}]$) should be calculated for the worst-case, which could be determined before the chip manufacturing by estimating the process variation impact. Also, as the VTCs in Fig. 5 show, since the state change of the positive output of inverter-based neuron is smoother, the steps of the β sweep could be larger compared to the negative one. Hence, in this work, we consider two different sweep steps (resolution) for negative and positive outputs of neurons, which are denoted by α_n and α_p, respectively. Note that the resolution for sweeping the weights impacts the accuracy of the fitted hyperbolic tangent, and subsequently, the accuracy of the neural network. On the other hand, increasing the resolution leads to an elongation of the process of extracting the VTCs. In this regard, we should apply weights – to extract VTCs – as fewer as possible with acceptable accuracy loss. As an example, Fig. 6 shows the accuracy of the fitted functions to the positive and negative outputs of neurons in 90nm technology under different resolutions. Based on the reported accuracies, in this work, we consider $\alpha_p = 0.06$ and $\alpha_n = 0.004$ resolutions for sweeping the weights.

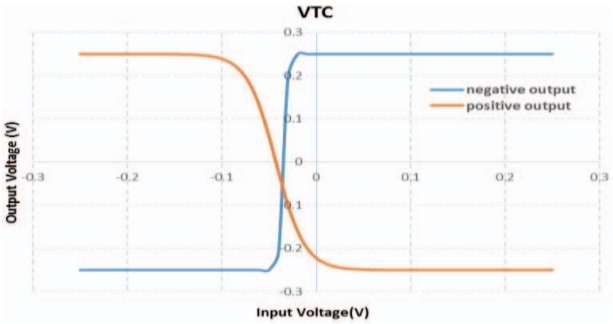

Fig. 5. VTCs of negative and positive outputs of the inverter-based neuron.

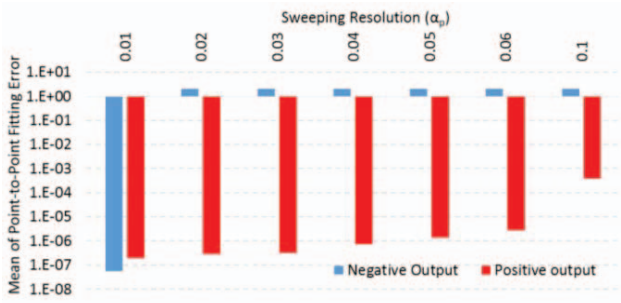

Fig. 6. Accuracy of fitted VTCs based on different sweeping resolutions. $\alpha_n = \alpha_p/2.5$.

In order to extract the activation functions of the neurons in hidden layers, except the ones in the first hidden layer, we use a similar approach, which is used in the output layer. However, because the output of these neurons are not directly observable, we should pass their outputs to the primary outputs of the network though a path of neurons generated by applying proper weights. To pass the output of a neuron (*e.g.*, a_j^i) to the output of a neuron in the next layer (*e.g.*, $a_{j'}^{i+1}$), the weights of all the incoming nodes of $a_{j'}^{i+1}$, except the one which comes from a_j^i should be set to zero, while the one which comes from a_j^i (*i.e.*, $w_{jj'}^{i+1}$) should be set to one.

As an example, Fig. 4(b), depicts application of the proposed approach to a neuron in hidden layer. In this example, the output of the i^{th} node is $\sigma_k\left(\sigma_j\left(\sigma_i(w_b x_b)\right)\right)$. Based on this, before determining σ_i, σ_j should be determined. Considering this, to extract the VTC of neurons of each layer, the activation functions of neurons in the next layer have to be determined. Thus, for neurons in the hidden layer, we start from the neurons of the last hidden layer. In the proposed algorithm, the result of β sweeping is observed from the outputs of neurons in output layer. Therefore, for each neuron in hidden layers, we should follow a unique path to get to a neuron in the output layer. Consequently, in i^{th} hidden layer, we are able to extract the VTCs of ξ_i neurons simultaneously. ξ_i is defined by

$$\xi_i = \min(N_{h,i+1}, \dots N_{h,L}, N_O) \tag{4}$$

where N_O ($N_{h,i}$) denotes the number of the neurons in the output (i^{th} hidden) layer. Also, $N_{h,L}$ indicates the number of neurons in the last hidden layer. Note that there should not be any common nodes in the paths of nodes, whose VTCs are being extracted. As an example, Fig. 4(c) shows the paths for extracting the VTCs of two nodes in a hidden layer.

For deploying a new weight, resistance of the corresponding memristor should be changed. Changing the resistance of memristor is time consuming due to its writing difficulty [8]. Hence, for the first hidden layer owing to accessibility to their inputs, we suggest to sweep the input value (*e.g.*, a_i^1) instead of the weight. Thus, first, similar to the other hidden layers, a path from the neuron (whose VTC should be extracted, *e.g.*, j^{th} neuron) to a neuron of the output layer is assigned. Next, weight from an input (*e.g.*, w_{ij}^2) is set to one, and the weights of other inputs to the neuron are set to zero. Then, the a_i^1 is swept (similar to weight sweeping), and the output of the path is observed. Fig. 4(d) shows an example of sweeping of a neuron in first hidden layer.

Based on the above discussion, the number of times that the proposed approach accesses the chip to extract the VTCs of neurons is as follows:

- Number of accesses for extracting the VTCs of neurons in the output layer:

$$ACC_L = \left\lceil \frac{\beta_{max,p} - \beta_{min,p}}{\alpha_p} \right\rceil \tag{5}$$

  where, $\beta_{max,p}$ ($\beta_{min,p}$) is the maximum (minimum) considered value for the bias weight to determine the VTC of positive output of neuron.
- Number of accesses for extracting the VTCs of neurons in hidden layers

$$ACC_H = \sum_{i=1}^{n} \left\lceil \frac{N_{h,i}}{\xi_i} \right\rceil \times \left(\left\lceil \frac{\beta_{max,p} - \beta_{min,p}}{\alpha_p} \right\rceil + \left\lceil \frac{\beta_{max,n} - \beta_{min,n}}{\alpha_n} \right\rceil \right) \tag{6}$$

where $\beta_{max,n}$ ($\beta_{min,n}$) is the maximum (minimum) considered value for bias weight to determine the VTC of negative output of neuron. This formula includes the number of access for extracting the VTC of the neurons in the first layer where instead of weights, the inputs are swept.

Note that in each access a set of weights (inputs) are applied and the outputs of the network are observed.

B. Training

After determining neuron activation functions, the network should be trained and proper weights should be computed. Backpropagation is an algorithm which is widely used in training Classical NNs [2]. The backpropagation algorithm employs the gradient descent optimization approach to minimize the cost function and determine the network weights [4]. The cost function may be defined by:

$$C = \sum_{j} (a_j^L - E_j^r)^2 \tag{7}$$

where E_j^r is the expected output of the j^{th} neuron of the last layer. In conventional networks using backpropagation, the activation functions of all neurons are considered to be the same [4]. Considering a predefined activation function for all neurons is not applicable in current technologies, since process variation impacts the activation functions. Therefore, in this work we suggest to consider neurons with different activation functions in a neural network. The output of the j^{th} neuron in i^{th} layer in this case is defined by

$$a_j^i = \sigma_j^i(\sum_{k} w_{jk}^i a_k^{i-1} + \beta^i b^i) \tag{8}$$

where σ_j^i is the activation function of the j^{th} neuron in i^{th} layer. By using this model, the manufactured chip could be trained. Since the training should be performed for each manufactured chip individually, the test time increases significantly. In order to address the long test time issue, we propose to reduce the search space for calculating each synaptic weight to a bounded range around a pre-calculated nominal weight. This restriction speeds up training while it may decrease the quality of training. For restricting the weight values – during the training, we define the r function as follows:

$$r(w_{jk}^i) = V_{jk}^i \times \left(1 + \gamma \times \frac{1 - e^{-w_{jk}^i}}{1 + e^{-w_{jk}^i}}\right) \tag{9}$$

where V_{jk}^i is the nominal weight from the k^{th} neuron in the $(i-1)^{th}$layer to the j^{th} neuron in the i^{th} layer and γ is the restriction parameter ($\gamma \in [0, \infty)$) which is defined by the user. Increasing the parameter γ leads to larger search space and consequently, smaller accuracy loss. Note that in the proposed backpropagation algorithm, the initial weights for the training are the nominal weights. As an example, Fig. 7 shows the output of function r when the $V_{jk}^i$is 0.5 under different γ values.

To apply the proposed range bounding approach during the backpropagation algorithm, the output model of neuron (*i.e.*, (8)) is modified as follows:

$$a_j^i = \sigma_j^i(\sum_{k} r(w_{jk}^i) a_k^{i-1} + \beta^i b^i) \tag{10}$$

As stated above, the γ value impacts accuracy of the network as well as training runtime. To study the γ value impact, 1500 neural network samples for the FFT benchmark (from Axbench package[16]) in the presence of process variation were trained by employing the proposed training technique. The average of accuracies and runtimes are reported in the Fig. 8. As the results show, by decreasing the γ value, the training run-time is reduced at the cost of some accuracy loss. In the rest of this work, without loss of generality, we consider a value of 1.5 for parameter γ.

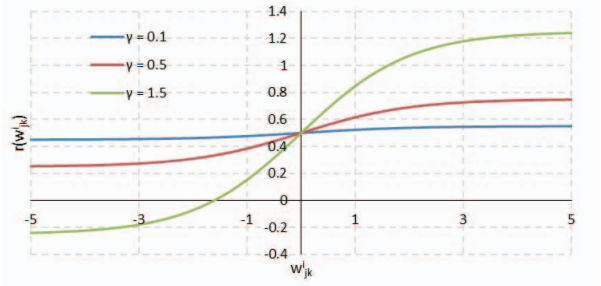

Fig. 7. Restriction function output for $V_{jk}^i = 0.5$ and for different value of γ.

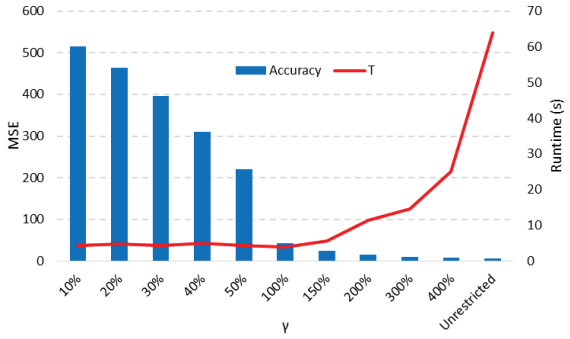

Fig. 8. The average accuracy and runtime of proposed training for FFT application under different γ values.

IV. Experimental Results

To assess the efficacy of the proposed approach, we have applied it on five benchmarks from the Axbench benchmark package. Table. I shows the type, training dataset, topology of their corresponding neural network, and the MSE of the trained network without considering the process variation. For modeling process variation, we assumed 5% normalized variation (*i.e.*, σ/μ) for the physical parameters of the transistors. The neurons were implemented in the 90nm CMOS technology. To model the impact of the process variation on their VTCs, HSPICE tool was employed. For each benchmark, we modeled 1500 chip samples. Note that the proposed restricted training algorithm was implemented by MATLAB tool.

Fig. 9 shows the accuracies of the neuromorphic chips of the studied applications by considering the impact of the process variation in two cases. In the first case, the nominal weights are applied while in the second case the proposed approach has been employed. As the results show, using the proposed approach leads to nearly 17X higher accuracy. Note that the process variation leads to lower accuracy, on average, by a factor of 308 compared to the case that nominal weight values were achieved

(see Table I and Figure. 9). However, by utilizing the proposed approach, the accuracy loss is decreased to a factor of 18 only, which proves the significant positive impact of the proposed approach on improving the accuracy.

Finally, Fig. 10 compares the number of accesses to the chips in the in-situ training approach and in the proposed approach. As the results show, the proposed approach, on average, leads to 123X fewer accesses to the chip. Furthermore, in every access to chip, the in-situ approach changes all of the network weights, whereas in our proposed approach, in most of cases, only up to N_O – number of the neurons in the output layer – weights are altered. Thus, the proposed approach is considerably faster than in-situ training.

TABLE I. THE EVALUATED BENCHMARKS, THE NETWORK TOPOLOGY AND THEIR NOMINAL ERROR IN THE ABSENCE OF PROCESS VARIATION.

Application	Type	Training Dataset	NN topology	MSE
FFT	Signal Processing	32768 Random Floating Point Numbers	1→8→2	0.4e-4
sobel	Image Processing	One 512×512 Pixel Color Image	9→8→1	1.9e-3
kmeans	Machine Learning	50000 Pairs of Random (r, g, b) Values	6→8→1	1.0e-4
blacksholes	Financial Analysis	16384 Data Point from PARSEC	6→6→1	2.8e-4
inversek2j	Robotics	10000 (x,y) Random Coordinates	2→12→2	1.11e-3

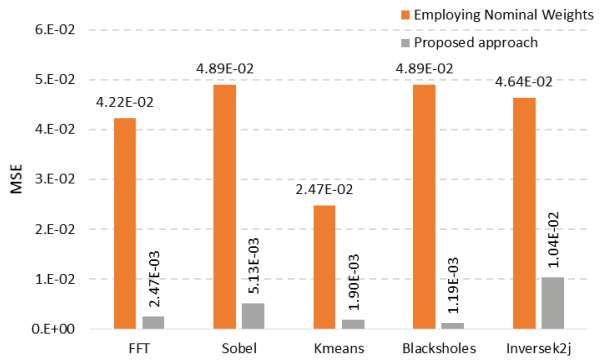

Fig. 9. The accuaracies of neuromorphic chips of different approximate applications in the presence of process variation under two approaches.

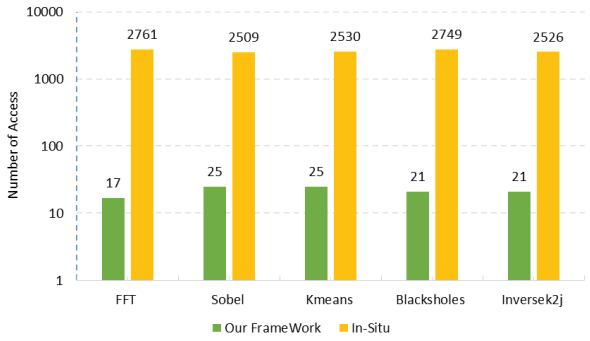

Figure 10. Number of access to chip in our proposed algorithm compared to the in-situe training approach under different approaximate applications.

V. CONCLUSION

In this paper, we proposed an approach to increase the accuracy of the inverter-based memristive neuromorphic circuits in the presence of process variation. This approach had two steps and applied after manufacturing the chip and during the test phase. In the first step of the proposed approach, the characteristics of the neurons have been extracted and in the next step, each neuromorphic chip has been trained by employing a fast and efficient ex-situ training. To evaluate the efficacy of the proposed approach, we have studied it on some approximate applications. The results revealed that this approach provides, on average, 123X smaller access to chip compared to the in-situ training.

VI. ACKNOWLEDGEMENT

The authors would like to thank Arash Fayyazi for helpful discussion.

REFERNCES

[1] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Technol. Intel Mag.*, vol. 9, no. 2, pp. 1–10, 2005.

[2] Z. Du *et al.*, "Neuromorphic accelerators: a comparison between neuroscience and machine-learning approaches," *Proceedings of 48th Int. Symp. Microarchitecture(MICRO)*, pp. 494–507, 2015.

[3] X. Liu *et al.*, "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," *Proceedings of 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[4] L. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., 1994.

[5] S. H. Jo *et al.,* "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[6] C. Yakopcic *et al.,* "Efficacy of memristive crossbars for neuromorphic processors," *Proceedings of Int. Joint Conf. on Neural Networks*, 2014, pp. 15–20.

[7] R. Hasan, C. Yakopcic, and T. M. Taha, "Ex-situ training of dense memristor crossbar for neuromorphic applications," *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2015, pp. 75–81.

[8] R. Hasan and T. M. Taha, "Enabling back propagation training of memristor crossbar neuromorphic processors," *Proceedings of Int. Joint Conference on Neural Networks*, 2014, pp. 21–28.

[9] Y. Xie and Y. Chen, "Statistical high-level synthesis under process variability," *IEEE Des. Test Comput.*, vol. 26, no. 4, pp. 78–87, 2009.

[10] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware Training for Memristor X-bar Beiye," *Proceedings of 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[11] D. Hunter *et al.*, "Selection of proper neural network sizes and architectures—a comparative study," *IEEE Trans. Ind. Informatics*, vol. 8, no. 2, pp. 228–240, 2012.

[12] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–8, 2013.

[13] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 75201, 2012.

[14] X. Guan, S. Yu, and H. S. P. Wong, "A SPICE compact model of metal oxide resistive switching memory with variations," *IEEE Electron Device Lett.*, vol. 33, no. 10, pp. 1405–1407, 2012.

[15] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," *Machine learning: From theory to applications*, Springer, 1993, pp. 9–28.

[16] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 449–460.