Accelerator-friendly Neural-network Training: Learning Variations and Defects in RRAM Crossbar

Lerong Chen[1], Jiawen Li[1], Yiran Chen[2], Qiuping Deng[3] Jiyuan Shen[1] Xiaoyao Liang[1] and Li Jiang[1]*
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[2]Department of Electrical and Computer Engineering, University of Pittsburgh, PA
[3]Lynmax Research, Beijing, China
Email: {clion0003, myloveys, shenjiyuan, liang-xy, ljiang_cs}@sjtu.edu.cn, yiran.chen@pitt.edu, dengqiuping@lynmaxtech.com

***Abstract*—RRAM crossbar consisting of memristor devices can naturally carry out the matrix-vector multiplication; it thereby has gained a great momentum as a highly energy-efficient accelerator for neuromorphic computing. The resistance variations and stuck-at faults in the memristor devices, however, dramatically degrade not only the chip yield, but also the classification accuracy of the neural-networks running on the RRAM crossbar. Existing hardware-based solutions cause enormous overhead and power consumption, while software-based solutions are less efficient in tolerating stuck-at faults and large variations. In this paper, we propose an accelerator-friendly neural-network training method, by leveraging the inherent self-healing capability of the neural-network, to prevent the large-weight synapses from being mapped to the abnormal memristors based on the fault/variation distribution in the RRAM crossbar. Experimental results show the proposed method can pull the classification accuracy (10%-45% loss in previous works) up close to ideal level with $\leq 1\%$ loss.**

I. Introduction

Neuromorphic computing using RRAM crossbar has gained a great momentum to achieve enormous energy efficiency [1]. The RRAM crossbar can take in the weighted combination of input signals (representing a vector) and naturally output the voltage representing the Dot-Product of a matrix-vector multiplication [2], [3]. Here, the resistances of memristors in the RRAM crossbar represent the matrix. For instance, a RRAM crossbar based neuromorphic design realizes a BSB training algorithm showing great potential to deliver incredible energy-efficiency with less hardware cost [4]. Xia et al. [5] design a novel RRAM crossbar architecture to minimize the AD/DA overhead and propose an optional ensemble method to boost the accuracy and robustness of the neuromorphic computing. Approximated results are normally allowed in these computation framework based the on memristor technology; it can achieve more than $20\times$ energy-efficiency with slightly degraded accuracy [1].

The memristor–a non-linear passive electrical component [6]–has been considered as the best way to realize the synapse of the neural-network thanks to its small size, non-volatility and low power consumption; its resistance represents the weight of the synapse (we simply use "weight" for clarity). Fig. 1 illustrates the structure of the 1R RRAM crossbar, wherein a memristor links the word-line and bit-line in the cross point. While 1T1R RRAM crossbar refers to the structure contains an access transistor and a memristor in the cross point. To deploy a neural-network on the RRAM crossbar, two memristors are used to represent the positive and negative weights, respectively. The memristor only stores the absolute value of the weight. The crossbar structure can provide a highly integrated cost-efficient solution for applications using neural-networks [4], [1], [5].

This work is partly supported by the National Natural Science Foundation of China (Grant No. 61602300, 61202026 and 61332001), Shanghai Science and Technology Committee (Grant No. 15YF1406000), Program of China National 1000 Young Talent Plan, NSF CNS-1253424, and a funding from Lynmax Research. *Li Jiang is the corresponding author.

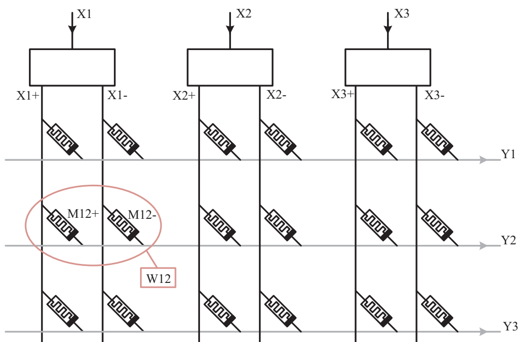

Fig. 1: The structure of a 1R RRAM crossbar.

Despite of these tremendous advantages, memristor suffers from various defects and variations, leading to dramatic yield degradation and causing significant error in neuromorphic computing. The process variations in the memristor device, on the one hand, cause deviation of the actual resistance from its ideal resistance [7], [8], [9]. On the other hand, the defects introduced in the fabrication process make the memristor stick to high/low resistance level [10]. The resulting RRAM crossbar may provide incorrect weights and finally bring significant classification errors to the output of the neural-network. The resistance variation in the memristor can be measured by programing the memristor to a target resistance state; the sensing circuit can then measure the actual resistance of all the memristors. After the measurement, the distribution of the resistance variation in a RRAM crossbar can be derived [11]. March-C algorithm [12], [13] and squeeze-search algorithm [10] are proposed to test the stuck-at faults in the memristor. To improve the test efficiency, Kannan et al. [14] intentionally summon sneak-paths in the RRAM crossbar to test multiple memristors at once. These test methods can pinpoint the exact location of abnormal memristor with stuck-at faults and resistance variations.

To tolerate the defects and variations, previous works have proposed various hardware-based solutions [15], [16], [8]. These solutions, however, brings inevitable hardware cost and power consumption. For 1T1R RRAM crossbar, we can remove the abnormal memristor by switching off the associated access transistor. However, it requires large routing overhead to control the individual access transistor; not to mention that the additional CMOS-based transistors bring significant hardware overhead and power consumption. Thus, the 1R RRAM crossbar is preferred [1], [5]. In this paper, we focus on tolerating the variation on the 1R RRAM crossbar. A software-based solution is proposed in [2] to mitigate the variation on memristor device. However, this method provides a general good solution for any RRAM crossbar, rather than an optimized solution for each specific RRAM crossbar based on the fault/variation distribution.

Therefore, in this paper, we propose a novel cross-layer solution leveraging the inherent self-healing capability of the neural networks. We first find an optimal mapping between weights and memristors (denoted as *weight-memristor*) based on a weighted bipartite-matching algorithm. Given the weight-memristor mapping, we propose to train the neural-network according to the distribution of the stuck-at faults (SAFs) and resistance variation in the RRAM crossbar, so that the weights with high influences will not be mapped to abnormal memristors. Compared to previous works, e.g., 10%-45% loss of accuracy is observed in [2], the proposed method can guarantee $\leq 1\%$ loss of accuracy; it can pull the accuracy from 6% to 70% even when significant variations occur.

The reminder of the paper is organized as follows: Section II introduces the related works and motivates this paper. Section III and IV describe the proposed matching algorithm and training method, respectively. Experiments are shown in section V. Section VI concludes this paper.

II. Prior works and motivation

In this section, we introduce the models and countermeasures of variations and faults in 1R RRAM crossbar and motivate this paper.

A. Variation Models and Fault Models in 1R RRAM crossbar

Memristors suffer from a wide range of variations which mainly fall into two categories: parametric variation and switching variation. The device-to-device parametric variation is caused by the imperfect fabrication, such as the line-edge roughness, the oxide thickness fluctuations and the random discrete dopants [17]. Consequently, the memristor has variable thickness and cross-section area, resulting in normalized accumulative resistance deviation [7]. The switching variation is a cycle-to-cycle variation caused by driving circuit. Any tiny fluctuations in the magnitude or pulse-width of the programmed current/voltage can lead to a large variation in the resistance of memristors (denoted as resistance variation) [9].

Other types of variations are well addressed in previous works, such as those that caused by the IR drops along the resistance network composed of metal wires and memristors [15] and the existence of sneak paths (unselected elements in the array form parallel paths across the device) [16].

The SAFs have been observed in a real RRAM crossbar chip [10]; they will cause low manufacturing yield of the RRAM crossbar chip. The stuck-at-zero (SA0) fault, on the one hand, is caused by over-forming defects, reset failure and short defects, which can affect 1.75% of the on-chip memristors [10]. A memristor device containing SA0 fault is always in the low resistance state (LRS). On the other hand, the broken word-line and the permanent open-switch defects force the 9.04% of the memristors to exhibit high resistance (HRS), denoted as the stuck-at-one (SA1) fault. A memristor in the LRS and HRS has 10k omh and 1M omh resistances, respectively. SA0 (SA1) faults can be clustered in a whole column (row) of the RRAM crossbar; they both can also be randomly distributed across the RRAM crossbar [10].

B. Variation Tolerance Methods

Hardware-based solutions are proposed to tolerate the above variations and SAFs. To eliminate the stochastic programming properties of memristors, in [8], constant reset pulses are repeatedly applied to a memristor until its resistance-level reaches the target range. System reduction schemes and IR-drop compensation techniques are proposed in [15] to resolve the physical limitation and the reliability issue induced by IR-drop. Two alternative crossbar architectures are proposed in [16] which can successfully eliminate the sneak-paths and provide better noise margin. Switching variation can be largely mitigated by adding additional reset pulses [8].

The drawback of the hardware-based solutions is the large hardware overhead and the high power consumption. Therefore, software-based methods are proposed to reduce the process variation on the RRAM crossbar. A general conversion algorithm is proposed [2] to map an arbitrary weight matrix of a neural-network to the conductance matrix of a RRAM crossbar. However, it pays no attention to the memristor variations and defects. Liu et al. [11] show a pre-calculation algorithm (denoted as VAT) that adjusts the training goal according to the impact of the variations; an adaptive mapping strategy is proposed to map the large weights to the memristors with low resistance variation. VAT is an off-device training method. It adds a scalar parameter γ in the training phase of the neural network to estimate the resistance variation of the RRAM crossbar. Repeatedly self-tuning of γ can derive new weight matrices. By testing the neural-networks defined by these derived weigh matrices, they find the optimal γ that obtains the maximum test rate. In a word, VAT tries to pre-calculate a set of weight matrices according to the priori significance of the variation; it then applies the best pre-trained weight matrix to a RRAM crossbar. Although VAT can find a generally good weight matrix for any RRAM crossbar, it is difficult to find the best weight matrix for a RRAM crossbar with specific fault distribution owning to the use of a global parameter γ. The test rate achieved by VAT can be low when the defect rate is or the significance of variation is high.

To alleviate the above problem, the same paper proposes an adaptive strategy to map the weight matrix to the resistance matrix of the RRAM crossbar. Rows in the weight matrix are exchanged to prevent the large weight from being mapped to the memristor with large variation. However, only rows are possible to be exchanged, which dramatically restrain the solution space. Our experiments show that the above techniques suffer from sharp reduction of the test rate when the memristor has severer resistance variation, or when the affected memristors spread along the row of the crossbar, which can be commonly observed as introduced in section II-A.

C. Motivation

Previous work [11] shows that an on-device solution is preferred owning to the hardware cost reduction. The timing cost of the neural-network training is ignorable as it is an one-time effort for each application. However, the test rate provided by existing solutions can still be significantly enhanced. Moreover, no software-based solution is provided to tolerate the SAFs.

An opportunity—ignored by all the existing solutions—for further improvement of the test rate is to explore the self-healing capability of the neural-network. In fact, the weight matrix of a neural network is always sparse, and the neural-network still works well with acceptable loss of information. Han et al. [18] propose to prune the near-zero weights and retrain the neural-network. This neural-network compression technique can dramatically reduce the storage of the weight matrix. They show that the neural-network can still work well after 80% of its weight are pruned. Inspired by the above work, it is a natural thought to reduce the weights mapped to the memristors with high resistance variations or SAFs; While after the training process, the neural-network itself can recover from the error induced by the changes of the weights. Therefore, in this work, we propose a novel neural-network training method which explores the self-healing capability of neural-network to tolerate the resistance variations and the SAFs in the RRAM crossbar.

III. A Basic Weight-Memristor Mapping Method

In this section, we describe a bipartite-matching based method to derive a weight-memristor mapping for variation and defect tolerance, which serves as the first step of the proposed accelerator-friendly neural-network training method described in the next section.

A. *Bipartite-matching method*

Considering a $n \times n$-column RRAM crossbar, its resistance matrix T and the corresponding weight matrix of the neural-network also has n columns. We first calculate the impact of the memristor variations by mapping the p^{th} row of the W to the q^{th} row of the T. The metric used in [6], i.e., "summed weighted variations (SWV)", is to measure the impact of variation in the weight-memristor mapping. We extend this metric to measure the impact of SAFs as follows:

$$SWV_{pq} = \begin{cases} \sum_{j=1}^{n} |w_{pj} - t_{qj}| & \text{for resistance variation} \\ \sum_{j=1}^{n} |w_{pj} - w_{max}| & \text{for SA0 fault} \\ \sum_{j=1}^{n} |w_{pj} - w_{min}| & \text{for SA1 fault} \end{cases} \tag{1}$$

wherein w_{pj} refers to the weight between neuron p and j in W, t_{qj} refers to the actual weight represented by the resistance of two memristors (connecting q to j) in T. Thus, $|w_{pj} - t_{qj}|$ shows the difference between the ideal weight and the actual weight. Our goal is to minimize the sum of SWV_{pq} for every row in the W.

In the greedy mapping algorithm [6], each row in W is iteratively mapped to one row in T to derive the smallest SWV. The chosen row in T is then removed from the candidate list to be further mapped. Depending on the scanning order of the row in W, this algorithm leads to a local optimal solution that cannot guarantee an overall minimized SWV. For example, Fig. 2(a) shows the mapping procedure of the greedy based method. According to the scanning order, the greedy mapping algorithm maps the first row in W—full of small weights—to the first row in T, which happens to contain all small variations. As the greedy algorithm continues, two more rows are mapped as shown in the figure. Unfortunately, the last row in T contains three memristors with high resistance variation, but it has to be mapped to the last row in W. In this case, large weights are assigned to the memristors with large resistance variations, resulting in a large SWV that may cause significant output error.

We discover that the weight-memristor mapping problem can actually have a weighed bipartite matching formulation: the rows in W and T are a set of vertices L_w and R_t, respectively. An edge $e \in E$ between a vertex v_l in L_w and another vertex v_r in R_t indicates the row v_l is mapped to the row v_r in the weight-memristor mapping between W and T. The weight on the edge is the $SWV_{v_l v_r}$ value when mapping v_l to v_r. The problem of finding a weight-memristor mapping with the minimal SWV is now transformed to the problem of finding the minimal-weight perfect-matching in a bipartite-graph $G(L, R, E)$. We adopt the "Kuhn-Munkres" (KM) algorithm [19] to derive the optimal solution in a polynomial time. The proposed bipartite-matching based method can also apply to the RRAM crossbar with redundant memristor rows [6]. Given R redundant rows in T, in stead of finding a minimum-weight perfect matching, the KM algorithm derives a minimum-weight maximum matching, which forms a mapping between the n rows of W and the n rows of T. The remaining R rows without matched are switched off and will not be used in the run time.

B. *Analysis of bipartite-matching based method*

Without any doubt, the bipartite-matching based algorithm can obtain a better weight-memristor mapping than the greedy-based

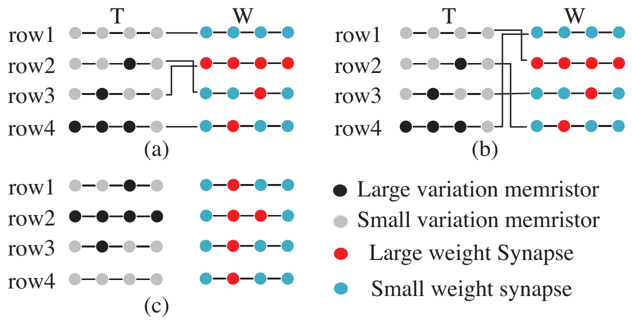

Fig. 2: Weight-memristor mapping examples that (a) the greedy-based method derives; (b) the bipartite-matching method derives; (c) the bipartite-matching method can hardly resolve.

method on a RRAM crossbar with small memristor resistance variation. However, the resistance variation may fluctuate sharply from one memristor to another; the memristors with high variations may cluster into a row/column. Similarly, the neural-network has sparse weight matrix, which in turn indicates the large weights can also be clustered. Fig. 2(c) shows such an example. In the second row of T, all the memristors have large resistance variations, while the third column in W is composed of large weights. However we change the mapping, a large weight is always mapped to a memristor with large resistance variation, which causes a large SWV and consequently decreases the test rate of the neural network .

The above example shown in Fig. 2(c) is supported by two facts: i) as mentioned in section II-A, SA0 and SA1 faults can spread over an entire row or column in the RRAM crossbar. ii) the biases[20] of a neural-network always contain important information; they occupy a whole column in the weight matrix; thereby, it is highly possible that a weight matrix can always have a column full of large weights. Therefore, we need a better solution to tolerate severer resistance variations and SAFs, as described in the next section.

IV. Accelerator-friendly Neural-network Training

All the mapping based solutions have a common weakness: the limited flexibility of the row-based matching narrows the solution space of finding the most fault/variation tolerable weight-memristor mapping. Based on the proposed bipartite-mapping algorithm, in this section, we explore the self-healing capability of the neural-network to enlarge the solution space of finding the fault tolerable weight-memristor mappings. The self-healing capability can be explained as follows: If several weights (synaptic connections) are removed from a neural-network, resulting in a loss of classification accuracy, the training process will make the surrounding weights compensate the above change, so that the whole neural-network bounces back to the original classification accuracy. This capability can be used to minimize the value of the weight mapped to a memristor with high variability or fault.

A. *Overflow*

The aim of the accelerator-friendly neural-network training is to minimize the SWV, by adjusting the weight matrix of the neural-network based on the distribution of faults and variations in the RRAM crossbar. Formally, given the weight matrix $W_{m\times n}$ derived from the bipartite-matching based method and the memristor resistance matrix $T_{m\times n}$, the proposed training method generates a new weight matrix $W^{'}_{m\times n}$.

Algorithm 1: The Algorithm for Accelerator-friendly Training.

```
   input  : Weight Matrix $W_{m\times n}$,Xbar_variation $T_{m\times n}$
   output: New Weight Matrix $W'_{m\times n}$
1  while Convergence $\neq$ TRUE || Test_rate is promoted do
2      Update SWV: $SWV_{m\times n} \leftarrow abs(T_{m\times n} \cdot W_{m\times n})$
3      $Max\_variation \leftarrow Find\_Max(SWV)$
4      if $SWV_{ij} = Max\_variation$ then
5          Reduce the weight (section IV-C): $W_{ij} \leftarrow W_{ij} \cdot t^{-1}$
6          Revise the Fix matrix (section IV-B): $F_{ij} \leftarrow 0$
7      end
8      //Retrain and test NN (section IV-B)
9      Retrain NN:$Train(W_{m\times n}, F_{m\times n}, Training\_sets)$;
10     Derive the Test_rate: Test_rate$\leftarrow$ $Test(W_{m\times n}, Test\_sets)$
11     Convergence $\leftarrow$ validate($W_{m\times n}$,$T_{m\times n}$,test_sets);
12 end
13 return $W_{m\times n}$;
```

As shown in Algorithm 1, the main procedure of the retraining method is to iteratively reduce the weight with the maximal SWV, and fix it in the follow-up training process (see line1-12). In each iteration, we first update the SWV according to the new weight matrix $W_{m\times n}$ and the resistance-variation matrix $T_{m\times n}$ (line 2) using equation 1. We then find one weight-memristor mapping with the largest SWV (line 3 and 4), e.g., SWV_{ij}. Next, we try to reduce the weight (W_{ij}) of the above mapping (line 5). It should be noted that we reduce the weight—using an exponential factor t—to $W_{ij} \cdot t^{-1}$ rather than to zero; the reason will be explained in section IV-C. However, the reduction of a weight in the neural-network leads to inaccurate calculation and results in the degradation of the classification accuracy. Thus, the neural-network should be trained again (the following steps are detailed in section IV-B). In short, we need to fix the updated weight W_{ij} using a Fix matrix $F_{m\times n}$, in which the coefficient (F_{ij}) corresponding to W_{ij} is set as zero (line 6). Afterwards, we train the neural-network again with a small learning rate (line 9); the small learning rate can result in a fine-grain self-healing rather than a disruptive change to the unfixed weights in $W_{m\times n}$. Consequently, the training algorithm can change the weights surrounding the fixed one in order to recover the classification accuracy of the neural-network. We validate the training process by testing the derived neural-network (line 10) and by checking the convergence of the training process (line 11). The iterative procedure goes on as long as the training process is not convergent or the test rate can still be promoted (line 1). Otherwise, the algorithm terminates and returns the new weight matrix (line 13).

B. *Fixing the weight in the training process*

We adopt the back-propagation method that utilizes the gradient descent technique to tune the weights. To show the training process, we use a two-layer fully-connected neural-network as an example. For a neuron j of output layer, its value y_j is calculated by:

$$y_j = f(\sum_m W_{ji} \cdot x_i(n)) \tag{2}$$

wherein n refers to the input vector consisting of a serial training samples, $x_i(n)$ refers to the input value of the neuron i in the input layer, W_{ji} refers to the weight connecting neuron i to j, m refers to the total number of neurons in the input layer associated with j, and f is the activation function.

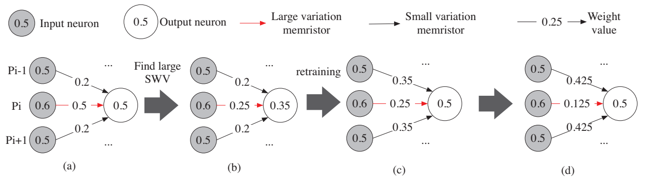

Fig. 3: Weight changing in the neural-network retraining method: (a) pre-trained weight; (b) fixing the weight connection; (c) after retraining; (d) in the next iteration.

In the training phase, the update of a weight W_{ji} can be derived as:

$$W_{ji} = W_{ji} - \alpha \cdot \delta_j(n) \cdot x_i(n) \cdot F_{ji} \tag{3}$$

wherein α refers to the learning rate, $\delta_j(n)$ is the local gradient value of neuron j, F_{ji} is the coefficient representing whether W_{ji} is fixed. The fix matrix F is used like this: if one weight W_{jk} is supposed to be fixed, we enforce $F_{jk} = 0$, and the result of $\alpha \cdot \delta_j(n) \cdot x_k(n)$ is ignored. Consequently, W_{ji} is not changed in the training process. Otherwise, for another input value $W_{ji} \cdot x_i(n)$ of neuron $j (i \neq k)$, the weight W_{ji} updates continuously until the convergence of the training process, using equation 3.

We now explain how the weights surrounding the fixed weight change in the training phase. In the back-propagation process, the significance of the gradient descent—how much the weight changes—associated with a neuron (e.g., j) is determined by input values from multiple neurons. When we reduce a weight (W_{ij}) connecting an input neuron i to j, an error occurs between the expected output of j and its real output; this error is induced by the miscalculation of i and W_{ij}. Fortunately, if other input neurons (e.g., $i-1$,$i+1$) are correlated with i, this error can be compensated by $i-1$ and $i+1$ in the back-propagation process: as the reduced weight W_{ij} is fixed in the back-propagation process, the gradient descent algorithm will instead change the weights connecting j to $i-1$ and $i+1$ to minimize the error. Generally, the values of the input samples, for a common neural-network based application, are highly correlated. For example, in a visual recognition application, the input data of the neural-networks are continuous because the image pixel at a nearby region have similar values.

Fig. 3 demonstrates the the above process. As shown in Fig. 3(a), a group of neighboring input neurons P_i, P_{i-1} and P_{i+1} have similar values. With the gradient descent algorithm, the associated weights also have similar values. When this pre-trained neural-network is mapped to a RRAM crossbar, the weight W_{P_iO} produces the largest SVW; thus, we reduce the weight by half and fix it, as shown in Fig. 3(b), which causes an increased error in the value of the output neuron. When this neural-network is trained again, the local gradient value of W_{P_iO}, due to the fixed W_{P_iO}, has to be embodied in $W_{P_{i-1}O}$ and $W_{P_{i+1}O}$ instead, so that the value of the output neuron bounces back to the original one, as shown in Fig. 3(c). This process continues, as shown in Fig. 3(d), until W_{P_iO} is too small to produce a countable SWV.

It should be noted that, we choose a small learning rate α to ensure the fine-grain tuning of the weight. Thus, the weights update and fluctuate with a narrow range in each iteration of the back-propagation process. It can avoid the disruptive change in a weight, e.g., a small weight mapped to a memristor with high variation suddenly changes to a larger one, which may lead to new pair of weight-memristor mapping with large SVW.

C. Reducing the weight

The way to reduce a single weight plays a critical role in proposed training process. It may cause a series of problems if we directly apply the weight-prune method used in Deep Compression [18]. Specifically, their method prunes the minimal weights in the weight matrix by setting the values of these weights as zero. Because of the sparse weight matrix, the neural-network has little loss of accuracy even when the majority of the weight are pruned. When applying the above method to the weight-memristor mapping—directly reducing the weight to zero—the test rate of the neural-network decreases sharply. Because the weight-prune method only needs to reduce a small weight; while we instead have to change a *large* weight, leading to a significant information loss. Even with the self-healing capability, the test rate can hardly bounce back to the same level of the original neural-network (running on a perfect RRAM crossbar). For instance, when a large weight is reduced to zero, its surrounding weights, probably also with small values, are updated to a large value for error compensation. It has a large likelihood that some of these surrounding weights are also mapped to memristors with large variation, resulting in large SVW. Consequently, they are chosen as the candidate for weight reduction in the next (or later) iterations in our training process. The resulting training process can hardly be convergent. Not to mention the fact that the surrounding weights of a large weight are likely to have large values. Moreover, if all the weights at a nearby region have been reduced to 0, some input values of the neural-network are mistakenly ignored, causing a danger of information loss. In some special cases, a single weight is more critical to the output than other weights. According to our experiments, if zero is assigned to a bias mapped to a memristor with large variation, the output calculated with this zero bias is too far away from the expected one; the resulting error can hardly be healed by the proposed training method. Therefore, in this work, we reduce the weight by an exponential coefficient t as follow:

$$W_{ij} \leftarrow W_{ij} \cdot t^{-1}; (t > 0) \tag{4}$$

We set the default value of t as 2 based on the experience from the experiments. Noted that a single weight can be reduced repeatedly, but the repetitions has an upper bound to prevent the loss of information. The upper bound can be different among different neural networks.

The remaining issue is the time consuming training processes to reduce many weights. Thus, we try to reduce multiple weights and fix them in a single training process. Specifically, we simultaneously reduce multiple "independent" weights which are far from each other. This strategy can prevent multiple correlated weights in a neighboring region from being reduced and fixed at once; otherwise, no surrounding weights can compensate the error. Meanwhile this strategy can reduce the number of training processes. Suppose in each iteration we reduce and fix N weights with the top N largest SWV at the same time. Table I shows a significant speedup—huge reduction of training iterations—of the proposed training method, with an ignorable drop of the test rate .

TABLE I: Reduce multiple weights in each iteration.

N	Number of iteration	Test rate
1	2000	89.7%
5	634	90.5%
10	378	90.3%
20	205	89.3%

V. Experiments

A. Experimental setup

To evaluate the efficiency of the proposed accelerator-friendly neural-network training method, we deploy a two-layer neural-network on a RRAM crossbar as the platform for digit recognition appellation. The inputs of the neural-network are the pixel values of the benchmark images; while the output signal is one of the ten Arabic numerals from 0 to 9. The RRAM crossbar contains two crossbar circuits; the differential result of two corresponding signals output from these two crossbars can represent either a positive or a negative value. The scale of each crossbar circuit is 784×10. The remaining set of parameters are the same as what are set in [11]. The experiments are tested with Monte-Carlo simulation method.

The initial classification accuracy of the neural-network for the MNIST data is about 90%, serving as the upper bound of all the variation/defects tolerant methods. Then, we set SAFs and resistance variations of memristors in the RRAM crossbar circuit. As the memristors with resistance variation will have abnormal resistances, they represent incorrect weights. The change of weight, from w_{pj} to $w^{'}_{qj}$, is shown as follows:

$$w^{'}_{qj} \leftarrow w_{pj} \cdot e^{\theta_{qj}}; \theta \sim N(0, \sigma^2) \tag{5}$$

wherein θ_{qj} represents the memristor resistance variation, which follows the lognormal distribution [8]. We use σ to present the significance of the variation in the experimental result. For stuck-at-zero (SA0) fault, the memristor is always at the LRS that represents a maximum weight; while for stuck-at-one (SA1) fault, the memristor gets stuck at HRS that represents a minimum weight. When we test the neural-network running on such RRAM crossbar circuit, we derive inaccurate classification results. We apply the proposed method and the one in [11] to the RRAM crossbar circuit and the derived classification accuracies are normalized to the reference accuracy (the upper bound) for comparison.

B. Results and analysis

Fig. 4(a) illustrates the test rate of various techniques only considering the resistance variation in RRAM crossbar. "Ideal" is the upper-bound; "before-map" is the one without any variation/fault tolerant techniques; "Vortex" is the hybrid solution in [11]; "bi-match" represents the proposed bipartite-matching based mapping method and "bi-retrain" represents the proposed accelerator-friendly training method on the basis of bipartite-matching method. The test rate decreases as the resistance variation becomes severer. It should be noted that the hybrid solution in Vortex is only applicable to a smaller range of the resistance variation as shown in the figure. Obviously, the hybrid solution dramatically outperforms the "before-map". The two proposed methods further outperform the "Vortex", especially when the resistance variation is large. The "bi-retrain" method can raise the test rate from 61.8% (only "bi-match") to 86.0% even under the significant variation $\sigma = 2$. Interestingly, the "bi-retrain" method is extremely close to the "ideal" test rate when no resistance variation and SAFs are injected in the RRAM crossbar.

Fig. 4(b) shows the test rate considering both resistance variations and SAFs. Solely applying the "bi-match" method only achieve 55.87% test rate in average; while the test rate of "bi-retrain" can reach as high as 89.27% and 71.02% when the resistance variation is small ($\sigma = 0.5$) and large ($\sigma = 2$), respectively. Compared to the ideal test rate, the "bi-retrain" method only has less than 5% loss of test rate in the "largest" variation set in [11], while the hybrid method in Vortex suffers 45% loss of test rate.

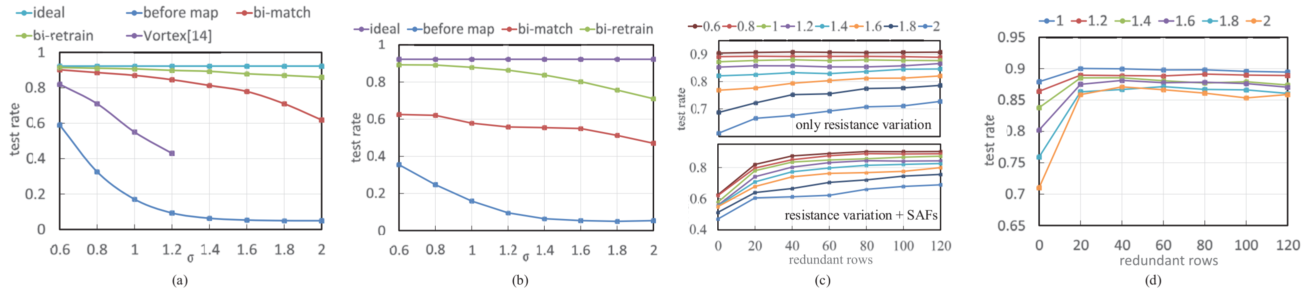

Fig. 4: Experiment Results on (a) test rate only considering the resistance variation by varying σ; (b) test rate considering the resistance variation and the SAFs by varying σ; (c) test rate for "bi-match" method by varying the redundancy amount; (d) test rate for "bi-retrain" method by varying the redundancy amount.

Fig. 4(c) shows the test rates derived by the proposed "bi-match" method after deploying various numbers of redundant rows in the RRAM crossbar. The test rates derived in different σ are shown in different color. The upper and lower figures show the test rate without and with SAFs, respectively. From both figures, we observe that the test rate hardly increases with more redundant rows when the resistance variation is small ($\sigma = 0.6 - 1.0$). In contrast, when the variation gets larger, the redundancy plays a more important role in improving the test rate. This observation also applies to the RRAM crossbar injected with SAFs. We find that the redundancy is very helpful to tolerate the SAFs when only "bi-match" method is adopted.

Fig. 4(d) show the test rate derived by "bi-retrain" method with redundant rows wherein both resistance variation and SAFs are considered. The results only show the test rate under large resistance variation. The "bi-retrain" method can keep the test rate above 85% with 20 redundant rows. Put it another way, the propose "bi-retrain" method can dramatically save the redundant cost to achieve the same test rate. When the number of the redundant rows is larger than 40, adding redundancy cannot improve the test rate anymore. Note that a weak fluctuation on test rate can be observed. This fluctuation naturally exists in the classification model of the neural-network .

VI. Conclusion

RRAM crossbar built on the basis of memristors has enormous energy-efficiency and thus is a promising platform for neuromorphic computing. However, the memristor devices suffer various process variations and defects, resulting in dramatic drop of the classification accuracy of neural-network based applications. Previous software-based methods can only tolerate small resistance variation and their efficiency degrades largely when the variation becomes significant. This paper proposes a novel off-device neural-network training method by judiciously explore the self-healing capability of neural-networks, which makes the neural-network more friendly to the RRAM crossbar based accelerators. The experiments show significant improvements of the proposed method in a large-scale RRAM crossbar with high resistance variations and clustered SAFs. It can achieve a near-ideal test rate with the maximum resistance variation considered in previous work. Future work will investigate the potential of the proposed training method in deep nerual-networks.

References

[1] B. Li et al. Memristor-based approximated computation. In *IEEE International Symposium on Low Power Electronics and Design*, pages 242–247, 2013.

[2] M. Hu et al. Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication. In *Design Automation Conference*, pages 1–6, 2016.

[3] L. Xia et al. Switched by input: Power efficient structure for rram-based convolutional neural network. In *Design Automation Conference*, pages 1–6, 2016.

[4] M. Hu. Bsb training scheme implementation on memristor-based circuit. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 80–87, 2013.

[5] B. Li et al. Merging the interface: Power, area and accuracy cooptimization for rram crossbar-based mixed-signal computing system. In *Design Automation Conference*, pages 1–6, 2015.

[6] D. B. Strukov et al. The missing memristor found. *Nature*, 453:80–83, 2008.

[7] D. Niu et al. Impact of process variations on emerging memristor. In *Design Automation Conference*, pages 877–882, 2010.

[8] S. R. Lee et al. Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory. In *Symposium on VLSI Technology (VLSIT)*, pages 71–72, 2012.

[9] S. Yu, Y. Wu, and H. S. P. Wong. Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory. *Applied Physics Letters*, 98(10):103514–103514–3, 2011.

[10] C. Y. Chen et al. Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Transactions on Computers*, 64(1):180–190, 2015.

[11] B. Liu et al. Vortex: variation-aware training for memristor x-bar. In *Design Automation Conference*, pages 1–6, 2015.

[12] A. J. Van De Goor and Y. Zorian. Effective march algorithms for testing single-order addressed memories. *Journal of Electronic Testing*, 5(4):337–345, 1994.

[13] Y. X. Chen and J. F. Li. Fault modeling and testing of 1t1r memristor memories. In *VLSI Test Symposium (VTS)*, pages 1–6, 2015.

[14] S. Kannan et al. Modeling, detection, and diagnosis of faults in multilevel memristor memories. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):822–834, 2015.

[15] B. Liu et al. Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems. In *International Conference on Computer-Aided Design*, pages 63–70, 2014.

[16] H. Manem et al. Design considerations for variation tolerant multilevel cmos/nano memristor memory. In *Symposium on Great lakes symposium on VLSI*, pages 287–292, 2010.

[17] A. Asenov et al. Intrinsic parameter fluctuations in decananometer mosfets introduced by gate line edge roughness. *Transactions on Electron Devices*, 50(5):1254–1260, 2003.

[18] S. Han et al. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015.

[19] H. W. Kuhn et al. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[20] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.