Fault-Tolerant Training with On-Line Fault Detection for RRAM-Based Neural Computing Systems

Lixue Xia[1], Mengyun Liu[2], Xuefei Ning[1], Krishnendu Chakrabarty[2], Yu Wang[1]
[1] Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing, China
[2] Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
e-mail: yu-wang@mail.tsinghua.edu.cn

ABSTRACT

An RRAM-based computing system (RCS) is an attractive hardware platform for implementing neural computing algorithms. On-line training for RCS enables hardware-based learning for a given application and reduces the additional error caused by device parameter variations. However, a high occurrence rate of hard faults due to immature fabrication processes and limited write endurance restrict the applicability of on-line training for RCS. We propose a fault-tolerant on-line training method that alternates between a fault-detection phase and a fault-tolerant training phase. In the fault-detection phase, a quiescent-voltage comparison method is utilized. In the training phase, a threshold-training method and a re-mapping scheme is proposed. Our results show that, compared to neural computing without fault tolerance, the recognition accuracy for the Cifar-10 dataset improves from 37% to 83% when using low-endurance RRAM cells, and from 63% to 76% when using RRAM cells with high endurance but a high percentage of initial faults.

1. INTRODUCTION

Machine learning is now widely used in a variety of domains, and brain-inspired neural computing is considered to be one of the most powerful applications of machine learning. To ensure that neural network algorithms are feasible in practice, hardware implementations require high computing capability and energy efficiency. However, CMOS technology is faced with the bottlenecks of scaling limitations and the "memory wall" for von Neumann architectures [1]. Consequently, the energy efficiency gap between application requirements and hardware implementation continues to grow.

Emerging devices such as metal-oxide resistive random-access memory (RRAM) and its associated crossbar array structure provide the basis for a promising architecture for brain-inspired circuits and systems. By exploiting the crossbar structure, an RRAM-based computing system (RCS) can realize vector-matrix multiplication in analog form and reduce the computational complexity from O(n^2) to O(1). Moreover, since RRAM provides memory for data storage, RCS provides an attractive solution for computing-in-memory. This architecture eliminates the high data transportation overhead that is inherent to von Neumann architectures; therefore, it significantly boosts energy efficiency, particularly for neural computing applications [2].

RRAM faults may occur during both chip fabrication and data processing. RRAM faults can be classified as soft faults and hard faults [3]. For soft faults, the resistance of the RRAM cell can still be tuned, but the actual resistance is different from the expected value. Soft faults are caused by variations associated with fabrication techniques and write operations [4]. For hard faults, the resistance of an RRAM cell cannot be changed; this category includes the stuck-at-0 (SA0) and stuck-at-1 (SA1) faults caused by fabrication techniques [5] and limited endurance [6]. A traditional RCS relies on off-line training, which first trains a network based on RCS and then maps the trained network on to RRAM arrays. In such an approach, the recognition accuracy for the network application is limited by both soft and hard faults that occur in each RRAM cell [7].

One approach to tolerate RRAM faults in an RCS is to use an on-line training scheme, which trains a neural network using the output of the RCS [7]. This method can tolerate soft faults by utilizing the inherent fault tolerance capability in neural computing algorithms. However, hard faults still limit the performance of on-line training due to the following reasons: 1) fabrication defects cannot be adequately tolerated by on-line training because the training procedure always attempts to tune some RRAM cells with a SA0/SA1 fault caused by fabrication, and 2) the limited write endurance of RRAM cells can lead to an increasing number of RRAM cells with a SA0/SA1 fault during the training procedure, and thereby result in decreased accuracy of RCS. Since we need to train another network in order to change the neural computing applications for RCS, most cells will become faulty after repeated write operations.

Researchers have shown that more than 50% of the weights in neural networks can be fixed to zero [8], which motivates us to further use neural algorithms to tolerate not only soft faults but also hard faults on RRAM. Two challenges arise in fault-tolerant on-line training in the presence of hard faults. First, we need to know the distribution of faults in RRAM cells during training. The test time of traditional test methods increases quadratically with the number of rows (columns) of the RRAM crossbar [9], and this is a key limiter for on-line testing. Second, traditional redundancy-based methods for memory design cannot be utilized to target hard faults in an RCS. This is because the basic unit of an RCS is an entire RRAM column rather than a single RRAM cell, and redundant columns may also contain, as well as, give rise to hard faults.

In this paper, we present a complete design flow that includes both on-line fault detection and fault-tolerant training for RCS. The main contributions of this paper are as follows:

1. We propose a fault-tolerant training flow with on-line fault detection for an RRAM-based neural computing system. In

DAC '17, June 18-22, 2017, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/3061639.3062248

this flow, the system can periodically detect the current distribution of faults and tolerate the faults in the next training phase using the inherent fault-tolerant capability of the neural network.

2. We propose an efficient on-line fault detection method by using quiescent-voltage comparisons. Modulo operations are utilized to reduce the hardware overhead. The proposed method can detect faults with more than 70% accuracy and more than 87% fault coverage within an acceptable test time.

3. We propose a threshold-training method and a re-mapping scheme during the training phase to tolerate faults using the sparsity of neural networks. Our results show that the accuracy can be restored to the original training accuracy for a fault-free scenario.

2. PRELIMINARIES AND MOTIVATION

2.1 RRAM-based Neural Computing

An RRAM cell is a passive two-port element with variable resistance, and multiple cells are used to construct a crossbar structure. Researchers have found that if we store a "matrix" on the conductances of RRAM cells in the crossbar structure and input a "vector" as input voltage signals, the RRAM crossbar is able to perform high-efficiency matrix-vector multiplication in analog mode. Specifically, the relationship between the input and output voltages can be expressed as: $i_{out,k} = \sum_{j=1}^{N} g_{k,j} \cdot v_{in,j}$, where $\vec{v}_{in}$ is the input voltage vector (denoted by $j = 1, 2, ..., N$), $\vec{i}_{out}$ is the output current vector (denoted by $k = 1, 2, ..., M$), and $g_{k,j}$ is the conductance matrix of the RRAM cells representing the matrix data [10]. Based on this structure, several RCS designs have been proposed, e.g., RRAM-based neural networks with fully-connected (FC) layers [10], and RRAM-based convolutional neural networks (CNN) with both convolutional (Conv) layers and FC layers [11].

2.2 Related Prior Work

To tolerate soft faults, researchers have demonstrated on-line training schemes for RRAM and other emerging non-volatile memory (NVM)-based neural networks. Strukov et al. implemented a 9×3 single-layer RRAM-based network by on-line training [7]. This work shows the benefit of on-line training to solve the mapping variations. However, it cannot target hard faults and the endurance problem due to the small size of the array and the simplicity of the application with only a few write operations.

For on-line fault-tolerant training on a larger network, we also need to know the distribution of faults. Recent work on fault distribution in the RCS are mainly focused on off-line fault detections [9, 12]. These designs can cover many RRAM fault types, and provide high fault coverage. However, the high time complexity limits their usefulness for on-line testing. Moreover, [12] does not support fault localization, and [9] requires sequential March diagnosis to determine fault locations. In [13], a parallel test method was proposed to reduce testing time, but this design is not scalable for large crossbars.

2.3 Motivational Example

We simulate the training procedure of the VGG-11 network on the Cifar-10 dataset [14] for both fault-free training and on-line training with hard faults. The percentage of RRAM cells with stack-as faults after fabrication is approximately 10% [5]. The endurance model is based on the published data that the endurances of cells obey a Gaussian distribution [3], and we set the mean endurance of cells to be equal to 5×10^6 [6, 15, 16]. The results are shown in Fig. 1. Training in the absence of faults can classify the $TestSet$ with 85.2% accuracy after 5 million iterations. However, when SA1 and SA0 faults are injected during training according to the above endurance model, the maximum accuracy can reach only 37%, and the accuracy further decreases if we continue with more training iterations. Moreover, if we need to train the RCS for another subsequent neural-computing application, which aggravates the impact of faults (i.e., the percentage of RRAM cells with faults may become 50%), we cannot achieve higher than 10% accuracy after training. Therefore, a fault-tolerant on-line training solution for RCS is required.

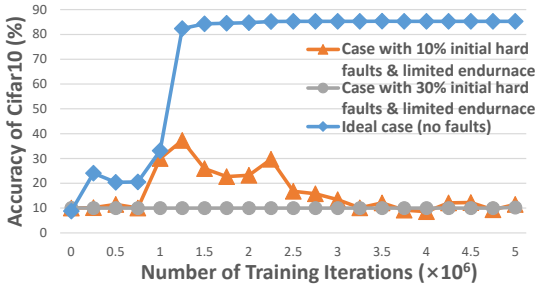


Figure 1: Training accuracy versus different initial hard fault conditions of RCS on the Cifar-10 dataset.

3. FAULT-TOLERANT TRAINING

The complete flow of the proposed fault-tolerant on-line training method is shown in Fig. 2. In priori work, a forward propagation phase is processed to obtain the actual output from the current neural network [7]. Then, the ideal output from the training dataset is compared with the actual output, and the difference is back-propagated to update network weights. In the proposed fault-tolerant training method, a threshold-training step (described in Section 5.1) is proposed after the back-propagation phase to enhance the lifetime by eliminating write operations on RRAM cells. Subsequently, after every fixed number of iterations, the proposed on-line fault detection method (described in Section 4) is executed to update fault-free/faulty status of RRAM cells. A pruning step is carried out simultaneously to obtain the locations of zeros in the weight matrices based on network pruning [8]. With the knowledge from the two distributions, the proposed re-mapping technique (described in Section 5.2) is applied to tolerate the faults by utilizing the inherent sparsity of neural algorithms.

4. ON-LINE FAULT DETECTION

4.1 Detection of Faults by Quiescent-Voltage Comparison

The flow of the proposed quiescent-voltage comparison method is described in Fig. 3. A read operation is first carried out to extract the RRAM crossbar values after training and store them off-chip. The SA0 and SA1 faults need separate detection procedures. The SA0 fault detection procedure consists of four steps. The first step is to write a fixed increment to all RRAM cells. We replace the traditional "Write 1" operation with a "Write $+\delta w$" operation, where $+\delta w$ is the fixed increment. In this way, we can recover the training weights of RRAM cells after testing. Second, test voltages are applied to one group of rows. Utilizing the crossbar structure, output voltages can be obtained at all column output ports concurrently. This parallel test method is therefore time-efficient. In the third step, control logic and a multiplexer are utilized to select an appropriate reference voltage. The reference voltages are set to be equal to the sum of previous values stored off-chip and the written

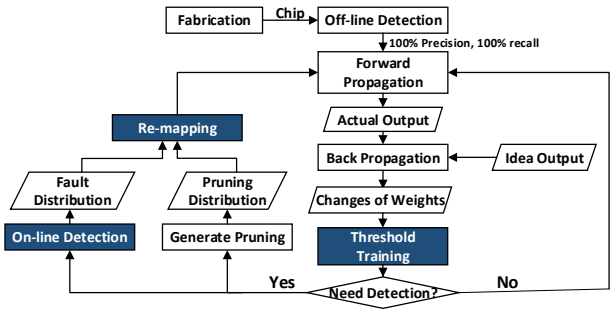


Figure 2: Proposed fault-tolerant training method with (1) a threshold-training method to reduce the write workload, (2) an on-line fault detection phase to detect fault locations, and (3) a re-mapping phase to avoid or reuse faulty cells.

increments. The last step involves comparisons between the actual outputs and the reference voltages. If a discrepancy exists, it denotes that at least one of the RRAM cell in the selected rows and columns cannot be updated correctly when we write an increment. This discrepancy indicates a SA0 fault. Because RRAM crossbars can be used in two directions, we can also apply test voltages to column input ports, and repeat the last three steps to derive the row information from the row output ports. After SA0 fault detection, a "Write $-\delta w$" operation is carried out for SA1 fault detection. The reduction value is set equal to the increment value used for SA0 detection to enable the RRAM to recover its original training weights. The four-step test process for SA1 fault detection is similar to that for SA0 fault detection.

An example is presented in Fig. 4(a) to illustrate the test procedure. The crossbar here is a simplified schematic of a 10×10 RCS, and a square with a cross represents an actual defective cell. The test size is set to 5, which implies that the test voltages are applied to five rows in every test cycle. Therefore, to finish the row test, we need two test cycles. An additional two cycles for column test are required. Shaded squares are determined as being fault-free. In Fig. 4(a), the white squares denote the detected fault locations. As shown in this figure, 100% fault coverage is obtained, but some fault-free cells are determined to be faulty.

4.2 Reduction of Hardware Overhead with Modulo Operations

To tolerate write variance, RRAM cells in the test phase of RCS can be considered as multi-level resistance cells. The increment (reduction) written in testing phase is set to be larger than the variance. The number of resistance levels can be set to 8 [17].

Although the same voltage is applied to all the tested rows, the correct outputs may still be in a large range for different conductance (g_{kj}) combinations. To simplify the design of the reference voltage, we use modulo operations to map all possible voltages to a limited number of values. We choose 16 as the divisor for the modulo operations based on a trade-off between fault coverage and hardware overhead. In this way, only 16 reference voltages are needed, and the control logic is also simplified. Faults can still be detected unless 16 or more faults occur simultaneously in the tested area. As the divisor for the modulo operation increases, the fault coverage increases, but the hardware overhead increases. In order to apply modulo operations, we reuse the analog-to-digital converters (ADCs) on the output ports, transforming the analog output voltages to digital values. The $\mod(2^n)$ operations can be realized by truncating the last n bits of the dividend. A set of NAND gates can then be used for digital comparisons. The loss in information due to the modulo operations leads to a slight decrease in fault coverage.

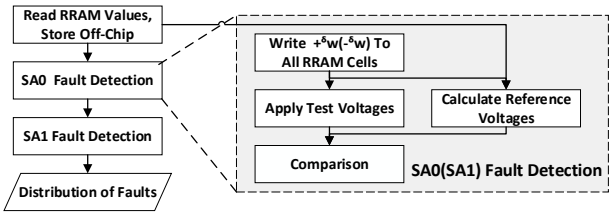
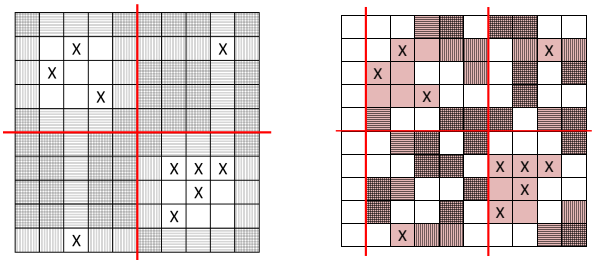


Figure 3: Description of the quiescent-voltage comparison method.



(a) Detect faults among all cells (b) Detect faults among selected cells

Figure 4: Illustration of the quiescent-voltage comparison test method (test size = 5).

4.3 Improvement of Performance by Selected-Cell Testing

To make the test method practical, we must ensure low test time, high fault coverage, and low false positives for fault detection. Using the quiescent-voltage comparison method for only the selected cells helps us to achieve this goal. Cells that need testing are selected based on the knowledge that SA0 faults can occur only in cells with high resistance, and SA1 faults can occur only in cells with low resistance. The read operation at the beginning of the test phase provides the necessary information about resistance values. The test procedure involving selected cells is shown in Fig. 4(b). For example, considering the detection of SA0 faults, the pink squares denote the cells with high resistance. Since SA0 faults can occur only in these pink cells, there is no need to test the other cells. In this way, the test time may be reduced because no test is applied to the first column. The number of false detections is also decreased from 10 in Fig. 4(a) to 6 in Fig. 4(b).

5. ON-LINE TRAINING

5.1 Threshold Training

We first introduce the threshold-training method to reduce the number of write operations in each training iteration. Specifically, a training iteration contains two steps [8]: a forward-propagation step to obtain classification results using the current network and a back-propagation step to adjust the weights. For a specific weight $w^n_{i,j}(t)$ in the ith row and jth column of layer n, where t is the iteration count, the next weight $w^n_{i,j}(t+1)$ is determined by:

$$w^n_{i,j}(t+1) = w^n_{i,j}(t) + \mathcal{LR} * \delta w^n_{i,j}(t) \quad (1)$$

$$\delta w^n_{i,j}(t) = x^n_i(t) * f'(y^n_j(t)) * \delta y^n_i(t) \quad (2)$$

where $x^n_i(t)$ is the input of $w^n_{i,j}$ from the previous layer in the forward-propagation step, $f'(y^n_j(t))$ is the derivative of the output in the forward-propagation step, and $\delta y^n_i(t)$ is the difference propagated from the next layer in the back-propagation step. $\mathcal{LR}$ is the learning-rate parameter used to control the training speed, which is first set to a large value and gradually decreased during training.

To reduce the occurrence of faults, we analyze the distribution of

Algorithm 1: Threshold-training Algorithm

Input: $Current\_\{w^n_{i,j}\}$, $\{WriteAmount^n_{i,j}\}$, $LearnRate$, $\overrightarrow{Sample_s}$,$\overrightarrow{IdealOutput_s}$
Output: $Next\_\{w^n_{i,j}\}$

1 $\overrightarrow{ActualOutput_s}$ = ForwardPropagation($\{Current\_w^n_{i,j}\}$, $\overrightarrow{Sample_s}$);
2 $\delta\vec{y}^N = \overrightarrow{IdealOutput} - \overrightarrow{ActualOutput}$;
3 Calculate $\{\delta w^n_{i,j}\}$ according to Equ.(2) layer by layer;
4 **for** $n = 1 : N$ **do**
5 **for** i, j **do**
6 **if** $\{\delta w^n_{i,j}\} < CalculateThreshold(WriteAmount^n_{i,j})$ **then**
7 $\{\delta w^n_{i,j}\} = 0$; $Next\_w^n_{i,j} = Current\_w^n_{i,j}$;
8 **end**
9 **else**
10 $WriteAmount^n_{i,j} = WriteAmount^n_{i,j} + 1$;
 $Next\_w^n_{i,j} = Current\_w^n_{i,j} + LearnRate * \{\delta w^n_{i,j}\}$
11 **end**
12 **end**
13 **end**
14 **return** $Next\_\{w^n_{i,j}\}$

δw among all the weights in one iteration. For approximately 90% of the weights, δw is less than $0.01\delta w_{\text{max}}$, where δw_{max} denotes the maximum δw in this iteration. When we consider the endurance problem, a small value of δw contributes only slightly to network training but reduces the lifetime of the RRAM cell.

Based on this observation, we only write to an RRAM cell with a large δw. The proposed threshold-training method is shown in Algorithm 1. Line 1 is the forward-propagation step processed on RRAM arrays, while lines 2-3 indicate the back-propagation step used to obtain δw. For lines 4-13, if δw is less than the training threshold, it will be reduced to zero. Therefore, the corresponding RRAM cell can avoid the write operation. The threshold is set to 0.01 of the maximum value of δw, which improves the average lifetime of each RRAM cell by approximately 15X.

From a software perspective, threshold training will enlarge the number of training iterations. We test the method on (1) a $784 \times 100 \times 10$ NN on MNIST dataset [18] and (2) the VGG-11 network on Cifar-10 dataset. Compared to the original training method, the number of training iterations is enlarged by 1.2X.

5.2 Fault-Tolerant Re-mapping Method

Although threshold training can reduce the impact of write operations, new hard faults induced by limited endurance will inevitably occur during the training procedure, and impact the network performance (i.e., the recognition accuracy).

At the software level, researchers have noted that the weights in a neural network contain a large amount of redundant information. Therefore, pruning methods have been proposed to fix more than 50% of the weight values to zero during training [8]. This finding motivates us to use the inherent sparsity of a neural network to tolerant the SA0 faults in an RRAM cell. This target can be achieved by re-ordering the columns/rows of the weight matrix, and map the zeros in the weight matrices to RRAM cells with SA0 faults.

The challenge in designing such an exchange method lies in the inherent connection between matrices. As shown in Fig. 5, from a software perspective, a neural network consists of multiple cascaded layers. Therefore, the outputs of the RRAM crossbar are connected to the inputs of another RRAM crossbar through peripheral neuron modules [2]. Consequently, if we independently exchange the rows or columns in each weight matrix with M neurons, an M-to-M routing module is required to connect different RRAM crossbars, which introduces high area and energy overhead. To address this problem, we only consider the re-ordering of neurons in this work. In other words, when the ith and jth columns of the $(n-1)$th layer's weight matrix are exchanged, the ith and jth rows of the nth layer will also be exchanged in a corresponding manner. From a software perspective, if we regard the neural network as a weighted graph, the re-ordering of neurons with connected weights will lead to an isomorphic network whose interconnection structure is the same as that of the original network.

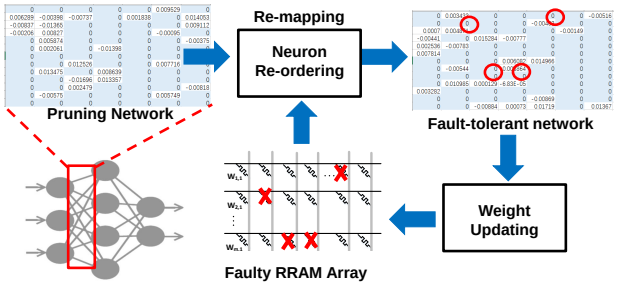


Figure 5: The proposed neuron re-ordering method for re-mapping.

After the fault-detection step, there are two networks: 1) an N-layer pruned network $\mathcal{P}$ provided by the pre-trained result from software training; 2) an N-layer fault distribution network $\mathcal{F}$, obtained after each fault detection phase described in Section 4. Specifically, $\mathcal{P} = \{P^{(1)}, P^{(2)}, ..., P^{(N)}\}$, where $P^{(n)} = \{p^{(n)}_{i,j}\}$ is a 2-dimensional weight-pruning matrix for the nth layer. If the weight in a neural network can be pruned, the corresponding value of $p^{(n)}_{i,j}$ is set to 0; otherwise, the value of $p^{(n)}_{i,j}$ is set as ∞. $\mathcal{F} = \{F^{(1)}, F^{(2)}, ..., F^{(N)}\}$, where $F^{(n)} = \{f^{(n)}_{i,j}\}$ is a 2-dimensional fault-distribution matrix for the nth layer. If an SA0 fault occurs on RRAM cell, the corresponding value of $f^{(n)}_{i,j}$ is set to 0; if an SA1 fault occurs, the corresponding value of $f^{(n)}_{i,j}$ is set to 1; otherwise, the value of $f^{(n)}_{i,j}$ is set as ∞.

To describe whether an SA0 fault in $\mathcal{F}$ is tolerated (reused) by the inherent zeros in the weight matrices, an $ErrorSet$ $\mathcal{E} = \{e\}$ is defined as the set of address groups, where $e = \{i, j, n\}$ is the address of a weight that satisfies: $e \in \mathcal{E}$ iff $(p^{(n)}_{i,j} \neq 0$ & $f^{(n)}_{i,j} \neq \infty)$. The optimization target is to minimize the number of elements of $\mathcal{E}$. Therefore, the distance between $\mathcal{P}$ and $\mathcal{F}$ can be defined as the number of elements of $\mathcal{E}$, i.e., $\text{Dist}(\mathcal{P}, \mathcal{F}) = |\mathcal{E}|$.

We can similarly define $E^{(n)}$ as the error set corresponding to layer n. The distance $\text{Dist}(\mathcal{P}, \mathcal{F})$ between $\mathcal{P}$ and $\mathcal{F}$ is the sum of distances between $P^{(n)}$ and $F^{(n)}$, i.e.,

$$\text{Dist}(\mathcal{P}, \mathcal{F}) = \sum_{n=1}^{N} \text{dist}(P^{(n)}, F^{(n)}) = \sum_{n=1}^{N} |E^{(n)}| \tag{3}$$

Based on these definitions, the optimization target of our neuron re-ordering algorithm can be described as follows. Given $\mathcal{F}$ and $\mathcal{P}$ with $(\|\mathcal{F}\| = \|\mathcal{P}\|)$, $\mathcal{O}$ is the set of the networks that $\mathcal{P}$ can be re-ordered into; the optimization result is to find $\mathcal{P}_{opt} \in \mathcal{O}$ that has the minimum distance from $\mathcal{F}$, i.e.,

$$\mathcal{P}_{opt} = \arg\min_{\mathcal{X} \in \mathcal{O}} \{Dist(\mathcal{X}, \mathcal{F})\} \tag{4}$$

The neuron re-ordering problem can be mapped to a set of K-napsack problems, hence it is NP-hard. We use a genetic algorithm to iteratively optimize the order of neurons layer by layer. For each layer, we randomly exchange two neurons and evaluate the change in the cost function $\text{Dist}(\mathcal{P}, \mathcal{F})$.

6. SIMULATION RESULTS

6.1 Evaluation Metrics for Fault Detection

We use the statistical metric of precision to evaluate false positives, where a false positive occurs when a fault-free cell is determined to be faulty. We also use the statistical metric of recall to evaluate test escapes. Higher the recall, lower is the test escape. Loss of precision results in unnecessary hardware overhead. Test escapes reduce the accuracy of neural computation. Let TP refer to the number of faulty cells that are correctly identified as being faulty. Let FP be the number of fault-free cells that are erroneously identified as being faulty. In addition, let FN be the number of faulty cells that are incorrectly identified as being fault-free. Then $Precision = TP/(TP+FP)$, and $Recall = TP/(TP+FN)$.

In addition to precision and recall, test time and test size are also important metrics. We define T_r (T_c) as the number of selected rows (columns) in each test cycle and C_r (C_c) as the number of rows (columns) in the crossbar. The test time, denoted by $\mathcal{T}$ and measured in cycles, is computed as: $\mathcal{T} = \lceil C_r/T_r \rceil + \lceil C_c/T_c \rceil$. We define E_r (E_c) as the number of rows (columns) that contain RRAM cells with the high (low) resistance; these cells are likely to be faulty cells. Since we need to perform test only in these selected rows and columns, the test time is reduced to: $\mathcal{T} = \lceil E_r/T_r \rceil + \lceil E_c/T_c \rceil$. In our simulations, without loss of generality, the number of rows and columns in the crossbar are assumed to be equal. We also set $T_r = T_c$ since the fault distributions are independent of the row or column directions.

6.2 Experimental Setup

6.2.1 *Fault Model*

Since there is no consensus yet on the spatial fault distribution in an RRAM crossbar, we evaluate our approach using several widely-used fault distributions [5] [19]. These include the uniform distribution and Gaussian distribution with several fault centers. The fabrication defects cause 10% of the RRAM cells to have stuck-at faults [5]. Two endurance models for RRAM cell are used to target both low-endurance RRAM cell and high-endurance RRAM cell. The mean endurance of low-endurance RRAM cell is set at 5×10^6 write operations and the distribution of endurance obeys a Gaussian distribution [3] with a variance of 1.5×10^6. For the high-endurance model, we assume that the distribution of endurance follows a Gaussian distribution with a mean of 10^8 and a variance of 3×10^7.

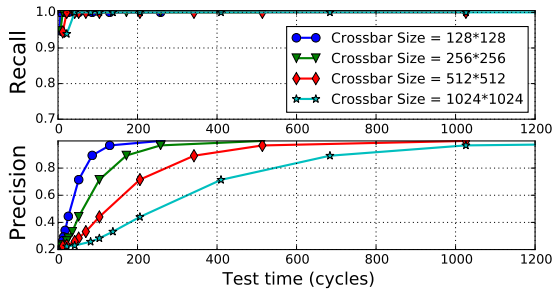
6.2.2 *Benchmarks*

We evaluate the proposed method on a modified VGG-11 deep neural network for the Cifar-10 dataset [14]. The VGG-11 network is modified to match the input size of Cifar-10 dataset, which contains 8 Conv layers and 3 FC layers. The total weight amount is 7.66M and the complexity is 137M operations. The recognition accuracy is 85.2% on $TestSet$, which is set as the ideal case for training without faults.

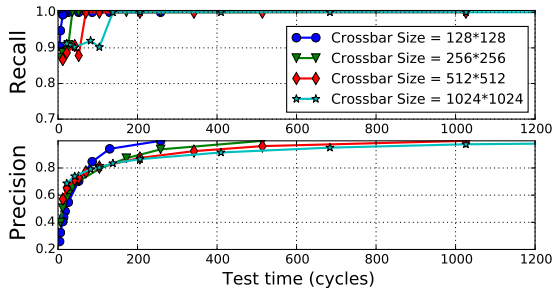
The RRAM crossbar sizes used in the fault detection experiments range from 128×128 to 1024×1024. This range is motivated by the current state of fabrication technology and the pro-spects for future development.

6.3 Effectiveness of Fault Detection

There are interesting trade-offs between test time, precision, and recall for the on-line test method. With a quiescent-voltage comparison using modulo operations and 10% of the cells being defective, the detection results are shown in Fig. 6(a) and Fig. 6(b) using the uniform and Gaussian fault distributions, respectively. The fault



(a) Uniform distribution cases



(b) Gaussian distribution cases

Figure 6: Trade-offs between test time, precision, and recall.

detection recall values increase slowly as the test time increases. Since the recall is always larger than 87%, it is always acceptable for the subsequent neural computation steps. On the other hand, for a given precision, the test time grows linearly with crossbar size. Therefore, large RRAM crossbars need longer test time to achieve satisfactory performance. However, even for a crossbar with 1024×1024 cells, a 74% precision and 91% recall can be obtained within 70 test cycles; thus, the test time is still acceptable for on-line test.

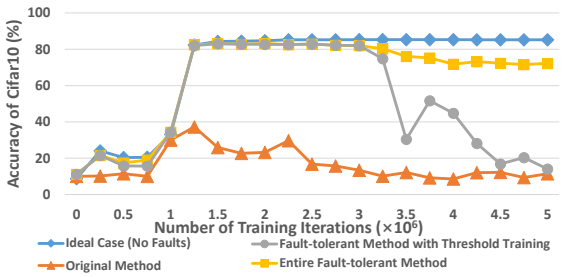
In order to demonstrate the advantage of testing only among selected cells, we compare the performance of testing among all the cells with the performance gain obtained by testing a subset of cells in proximate test time. Consider a Gaussian distribution of faults, and assume that 10% of the cells are faulty and 30% of the cells are in a high-resistance state. Our results show that, with this improvement, the precision increases significantly from around 50% to 77%, while the recall of both methods is maintained above 90%.

6.4 Results of Entire Fault-tolerant Training

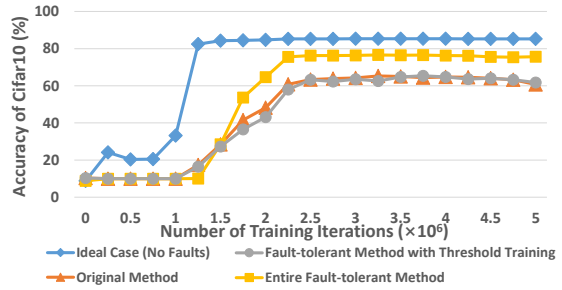
For RRAM-based CNN, some researchers implement both Conv layers and FC layers on RCS [20], while other researchers only implement FC layers on RCS [10]. Software level results have shown that the fault tolerance capability and sparsity of the Conv layer are much lower than that of the FC layer [8]. Therefore, we evaluate the proposed methods in two cases. (1) An entire-CNN case: all the layers of the VGG-11 network are mapped onto an RCS. (2) A FC-only case: only the FC layers are mapped onto an RCS.

We evaluate the original training method using multiple endurance models and initial fault ratios in both cases. Our simulation results show that Conv layers are sensitive to hard faults, and thus, the accuracy of CNN is approximately 10%, if more than 20% of the RRAM cells have hard faults. However, for the FC-only case, the accuracy decreases only when the percentage of faulty cells is larger than 50%.

Fig. 7(a) presents the training accuracy for the entire-CNN case. The mean endurance value of RRAM cells is set to 5×10^6. Without fault-tolerant training, the accuracy can drop to only 10%, and



(a) Entire-CNN case



(b) FC-only case

Figure 7: Training accuracy of fault-tolerant on-line training method with different endurance models on the Cifar-10 dataset.

the largest accuracy obtained during training is also lower than 40% (the orange line). The threshold-training method (the grey line) can increase the peak training accuracy to 83% on the Cifar-10 dataset, which is comparable to the fault-free software-based training results (the blue line). However, since the fault tolerance and sparsity of Conv layer is low, the proposed fault-detection and re-mapping steps cannot further improve the accuracy (the yellow line).

We further evaluate the effect of threshold-training method on the percentage of faulty cells for different endurance models. When a high-endurance model with mean value 10^8 is used, the original method can train the RCS for approximately 10 times. Since the average number of write operations in the threshold-training method can be reduced to only 6% of that of the baseline method, the threshold-training method can train the RCS for more than 150 times. For the endurance model with a lower mean value like 10^7, the percentage of faulty cells after the first training phase is approximately 14% using the original method; while in the second training phase, the training phase does not converge. The threshold-training method can successfully train the network with 14% of the cells being faulty; therefore, the RCS can be trained for approximately 27 times.

For the FC-only case, we focus on the scenario in which the RCS has been trained multiple times and contains a large number of initial faults before the subsequent training. We set the mean endurance value to 10^8, but reduce the remaining endurance of RRAM cells to mimic the case in which the RCS has already been trained multiple times. Fig. 7(b) shows the results for the case in which the percentage of RRAM cells with hard faults before training is approximately 50%. With such a large number of initial hard faults, the peak accuracy of traditional on-line training method (the orange line) is only 63%. The threshold-training method (the grey line) has negligible impact in this case because it can only reduce the occurrence rate of new faults but cannot tolerate the existing faults. The proposed fault-detection and re-mapping steps (the yellow line) can identify the hard faults of RRAM cells and tolerate them in the RCS utilizing the sparsity of neural-computing algorithms. Based on the proposed fault-tolerant on-line training flow, the accuracy can be increased back to 76%.

7. CONCLUSIONS

We have presented a fault-tolerant training method for an RRAM-based neural computing system to reduce the impact of hard faults in RRAM cells. An on-line fault-detection phase has been proposed to obtain the distribution of faulty cells during training by using a quiescent-voltage comparison method. In the training phase, threshold training and heuristic re-mapping based on neuron reordering are proposed to tolerate faults using the inherit sparsity of neural networks. Simulation results show that the accuracy in the presence of faults can be restored to the original training accuracy for a fault-free scenario.

8. ACKNOWLEDGEMENTS

This work was supported by 973 project 2013CB329000, and National Natural Science Foundation of China (No. 61373026, 61622403), Joint Fund of Equipment Pre Research and Ministry of Education, and Huawei.

9. REFERENCES

[1] M. M. Waldrop, "The chips are down for Moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[2] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA*.

[3] R. Degraeve *et al.*, "Causes and consequences of the stochastic aspect of filamentary RRAM," *Microelectronic Engineering*, vol. 147, pp. 171–175, 2015.

[4] L. Xia *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, 2016.

[5] C.-Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE TC*, vol. 64.

[6] K. Beckmann *et al.*, "Nanoscale hafnium oxide RRAM devices exhibit pulse dependent behavior and multi-level resistance capability," *MRS Advances*, pp. 1–6, 2016.

[7] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[8] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," *CoRR, abs/1510.00149*, vol. 2, 2015.

[9] S. Kannan *et al.*, "Modeling, detection, and diagnosis of faults in multilevel memristor memories," *IEEE TCAD*, vol. 34.

[10] L. Xia *et al.*, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," in *DATE*, pp. 469–474, 2016.

[11] T. Tang *et al.*, "Binary convolutional neural network on rram," in *ASP-DAC*, pp. 782–787, IEEE, 2017.

[12] S. Kannan *et al.*, "Sneak-path testing of memristor-based memories," in *VLSID*, pp. 386–391, IEEE, 2013.

[13] T. N. Kumar *et al.*, "Operational fault detection and monitoring of a memristor-based LUT," in *DATE*, pp. 429–434, IEEE, 2015.

[14] A. Torralba *et al.*, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE TPAMI*, vol. 30.

[15] C.-H. Cheng *et al.*, "Novel ultra-low power RRAM with good endurance and retention," in *VLSI Symp. Tech. Dig*, pp. 85–86, 2010.

[16] Y.-S. Fan *et al.*, "High endurance and multilevel operation in oxide semiconductor-based resistive RAM using thin-film transistor as a selector," *ECS Solid State Letters*, vol. 4, no. 9, pp. Q41–Q43, 2015.

[17] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *DAC*, pp. 1–6, IEEE, 2013.

[18] Y. LeCun *et al.*, "The MNIST database of handwritten digits," 1998.

[19] C. Stapper, "Simulation of spatial fault distributions for integrated circuit yield estimations," *IEEE TCAD*, vol. 8.

[20] L. Xia *et al.*, "Switched by input: power efficient structure for RRAM-based convolutional neural network," in *DAC*, p. 125, ACM, 2016.