Cost- and Dataset-free Stuck-at Fault Mitigation for ReRAM-based Deep Learning Accelerators

Giju Jung[1], Mohammed Fouda[2], Sugil Lee[1], Jongeun Lee[1], Ahmed Eltawil[2,3], Fadi Kurdahi[2]
[1]Dept. of Electrical Engineering, Ulsan National Institute of Science and Technology (UNIST), Ulsan, South Korea
[2]Center for Embedded & Cyber-physical Systems, University of California–Irvine, CA, USA
[3]CEMSE Division, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia.

***Abstract*—Resistive RAMs can implement extremely efficient matrix vector multiplication, drawing much attention for deep learning accelerator research. However, high fault rate is one of the fundamental challenges of ReRAM crossbar array-based deep learning accelerators. In this paper we propose a dataset-free, cost-free method to mitigate the impact of stuck-at faults in ReRAM crossbar arrays for deep learning applications. Our technique exploits the statistical properties of deep learning applications, hence complementary to previous hardware or algorithmic methods. Our experimental results using MNIST and CIFAR-10 datasets in binary networks demonstrate that our technique is very effective, both alone and together with previous methods, up to 20% fault rate, which is higher than the previous remapping methods. We also evaluate our method in the presence of other non-idealities such as variability and IR drop.**

***Keywords*—Artificial Neural Networks (ANNs), ReRAM Crossbar Array, Stuck-at Fault, Batch Normalization**

I. Introduction

Resistive random access memories, or ReRAMs, can provide high R_{off}-to-R_{on} ratio and high device density as well as extremely fast matrix-vector multiplication (MVM) capability via simple current summation on a crossbar structure. This unique set of features has motivated research into efficient ReRAM-based deep neural network (DNN) accelerators and optimization techniques [1]–[3]. On the other hand, ReRAMs in general suffer from non-ideal device behaviors such as low device reliability (i.e., bit failures), low endurance, and high variability/disturbance issues [4]. In particular, the reliability issue is very critical for today's ReRAM technology, with a typical fault rate being about 10% [5], [6], which is unacceptable for most applications.

Barring hardware redundancy approach [7], previous work on stuck-at fault (SAF) mitigation for ReRAM-based DNN accelerators can be classified into three categories: *retraining*, *correction*, and *remapping*. Retraining DNNs against faulty ReRAM Crossbar Architectures (RCAs) has been shown to be very effective [8], [9]. However, retraining must be done for each individual accelerator, which is not always feasible or desirable due to dataset availability, privacy, and/or high computation cost. Thus there is a need for low-cost, dataset-free fault-mitigation techniques for ReRAM-based DNN accelerators.

The *correction* approach [6], [10], [11] employs a post-processing step involving extra logic and memory, presumably in CMOS, to correct distorted MVM result. While this approach can guarantee correct result, it has very high cost because the post-processing step is essentially another, albeit sparse, MVM computation. The idea of *remapping* [8], [12], [13] is to change the mapping from weights to ReRAM cells, such that weight value '1' is mapped as much to stuck-at-one cells as possible and vice versa This requires varying amount of extra hardware depending on the method, and furthermore, has limited efficacy due to its limited remapping opportunities.

In this paper, we propose a novel approach called *free parameter tuning (FPT)*, which is similar to retraining (as it updates network parameters) but does not require datasets, additional hardware, gradient-based training, or any backward pass.[1] Unlike remapping, which is completely agnostic to the kind of computation being performed, our technique exploits the statistical property of DNN computation, and is hence complementary to remapping techniques. Our technique targets binary ReRAM devices as they are more mature and practical with minimal variability issues. Our work shows a high resilience of the training level even at a high fault rate of 20% and 40% in Binary Neural Networks (BNNs). FPT also produces better results in comparison with, and on top of, previous methods [8], [11], [13]. We also evaluate our method in the presence of other non-idealities such as variability and IR drop.

II. Related Work

A. SAF Mitigation Techniques

Previous work on the stuck-at fault mitigation in RCAs can be classified into three categories (see Table II). First is **retraining** [6], [8], [9], [12]. Retraining is simply to train the DNN again using a gradient-descent training algorithm while fixing some weight elements to constant values based on SAF information, which requires two things: a fault map [4], [5] and a training dataset. In addition, retraining has a high computational requirement, and must be done for each individual

This work was supported by NRF grants funded by MSIT of Korea (No. 2016M3A7B4909668, No. 2017R1D1A1B03033591, and No. 2020R1A2C2015066), IITP grant funded by MSIT of Korea (No.2020-0-01336, Artificial Intelligence Graduate School Program (UNIST)), and Free Innovative Research Fund of UNIST (1.170067.01). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.
J. Lee is the corresponding author of this paper (Email: jlee@unist.ac.kr).

[1]It is free in the dual sense of cost-free (i.e., no additional hardware) and dataset-free.

DNN chip. On the other hand, retraining can give a highest level of accuracy recovery with no hardware cost. In addition, there is no post-processing required after writing weights to devices after retraining, and for that reason, it is easy to link with other post-processing methods like remapping. Authors in [6] proposed knowledge distillation (KD)-based retraining that uses a teacher-student model. The target DNN, which is the student network, is trained to resemble the output of the teacher model, which may give better fault recovery.

Second is **correction** [6], [10], [11]. Authors in [10] proposed a simple technique where the result of distorted MVM due to a faulty ReRAM crossbar array can be corrected by post-processing. The post-processing step computes the contribution of faulty ReRAM devices toward MVM result, which is essentially another MVM, though it is done in a sparse fashion, with full access to the input vector and the faulty ReRAM cells. This post-processing is implemented in CMOS circuit to avoid another reliability issue, and gives very high accuracy recovery, but also has very high cost due to the added MVM operation, and needs a digital hardware module. The other techniques [6], [11] are very similar to [10], though RSA [6] aims to reduce the overhead of the correction step by exploiting weight importance.

The third category is **remapping**. Matrix permutation [8], [11]–[13] is a method of reshaping the matrix so that the value of the fault location matches the fault value as much as possible. It is divided into row permutation [8], [11], that requires an additional router, and neuron permutation [12], [13], that does not require a router. In the case of matrix permutation, there is a problem that a considerably larger preprocessing time is required as the network size increases in order to find the optimal permutation in common. In case of node permutation, the weight position must be reshaped if the convolution layer is performed with ReRAM crossbar arrays. Thus, there is a problem that it cannot be used in convolution layers and is limited to a model consisting only of a fully-connected layer.

The idea of row flipping [13] is to flip the sign value of the current row and shift the values of one row to be as close as possible to the SAF value. This method is fairly straightforward as it changes only the sign and does not require heavy post-processing, but it has a disadvantage that it may require a hardware module to obtain an input sum when shifting.

B. Weight Realization

Realizing a weight tensor as resistance matrices of RCAs is straightforward except that the size and value range of a weight tensor can exceed those of a RCA. The size problem can be handled by partitioning [14], then the output of RCAs needs to be summed before activation function. To handle negative weights one can add a constant offset so that all the weight values become non-negative. Implementing the offset in RCAs requires an extra column, the output of which needs to be subtracted from the RCA output, which is referred to as *unbalanced* weight realization [15]. Alternatively, a pair of RCA arrays (or ReRAM cells) can be used, so that subtracting the output of one (*negative array*) from that of the other (*positive array*) can implement signed weight values, which is called *balanced* weight realization [14]. For our experiments with BNNs, we use the weight realization schemes listed in Table I, where (x, y) in the balanced cases refers to a resistance value pair for positive/negative arrays.

TABLE I: Weight to resistance mapping. LRS and HRS refer to low and high resistance states, respectively.

Scheme	$+1$	-1
Unbalanced	LRS	HRS
Balanced	(LRS, HRS)	(HRS, LRS)

TABLE II: Classification and comparison of previous work

Category	Method	Downside
Retrain	Weight significance [8], [9]	Requires dataset
	Threshold train [12]	Requires dataset
	KD (Knowledge Dist.) [6]	Requires dataset
Correct	Output compensation [10], [11]	HW overhead, Extra calc.
	RSA [6]	HW overhead, Extra calc.
Remap	HW redundancy [7]	HW overhead (crossbar)
	Neuron permutation [12], [13]	Inapplicable to conv. layer
	Row permutation [8], [11]	HW overhead (router)
	Row flipping [13]	HW overhead (negation)

C. Batch Normalization

A batch normalization (BN) layer applies an affine transformation in (1) to the output of the preceding layer (x) which is either a convolution or fully-connected layer. It is known to reduce the internal covariate shift problem [16], and widely used to improve training performance. It has four parameters per output channel, which are updated during training only. Forward parameters (μ, σ) are the input statistics per batch updated in the forward phase, and backward parameters (β, γ) are updated through gradient descent in the backward phase (see Figure 1a). For inference, μ, σ are replaced with constant values, which may be obtained from the EMA (exponential moving average) of μ, σ during training (see Figure 1c). To further simplify computation, BN layers can be folded into the preceding layers, resulting in modified weight/bias values [17].

$$y = \gamma \left(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta \tag{1}$$

III. Free Parameter Tuning

A. Motivation

Optimization Perspective: Our initial motivation comes from the need to lower cost. The correction approach can achieve near 100% accuracy recovery given enough hardware resources, but is very costly. It would be ideal if we can recover accuracy by modifying biases only, but without explicit gradient back-propagation based training, since any gradient-based training would require a dataset and incur high computation cost. The fact that a BN layer can be folded into the preceding

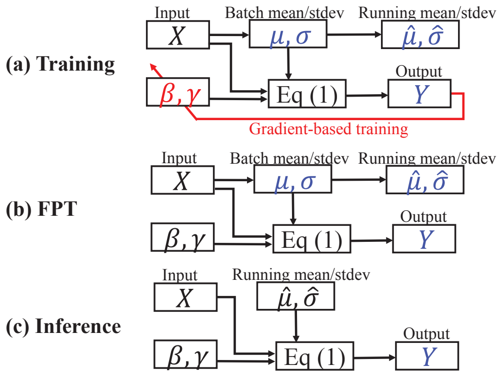
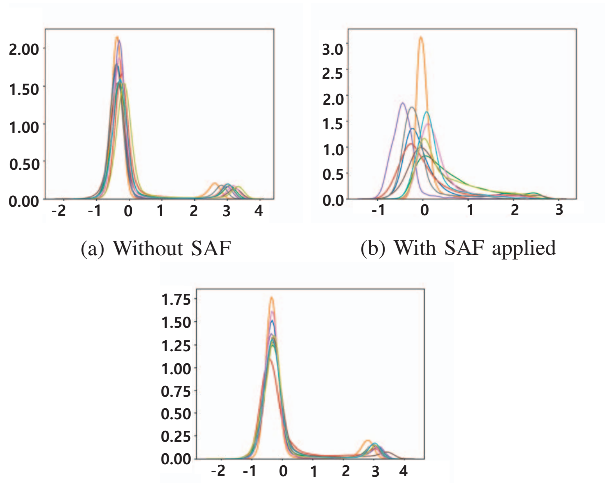


Fig. 1: Batch normalization layers run differently during training, FPT, and inference. Variables in red are updated by gradient descent, and those in blue are by the forward phase.



(a) Without SAF (b) With SAF applied

(c) With SAF and FPT applied

Fig. 2: Distribution of output activation (after BN) at the last layer before soft-max in the MNIST BNN, stuck-at-open 9%, stuck-at-close 1%. Colors mean different output neurons.

layer motivates the use of BN layers as an alternative to directly changing bias values.

Statistical Perspective: To see the effect of device defect, we plot the distribution of output activation with and without SAFs in Figure 2. In the absence of faults, output distribution is almost identical across neurons, with one large peak around zero and one small peak around 3. With faults, however, we observe large distortion in the graphs, and that the distortion patterns can vary among neurons (mean is shifted and standard deviation is increased). Thus one may expect to see improved accuracy if the distribution graphs are changed back to their original shapes, which is, coincidentally, one of the primary objectives of BN. Indeed we observe that after injecting SAFs to weights, just training BN layers can recover most of accuracy degradation due to SAFs (see Table III). Since

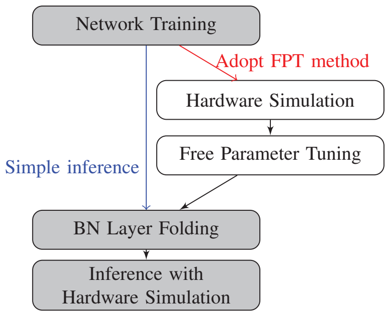


Fig. 3: Our experimental flow.

BN-only training is still a training, we explore a variation of BN-only training, which we call *Free Parameter Tuning*, in which only the forward parameters (μ, σ) are adjusted. Updating forward parameters is done during the forward phase and does not require back propagation or a full dataset. Hence, FPT represents a computationally cheap operation that is free of dataset or extra hardware.[2]

B. Detailed Method

Figure 3 shows two flows. When not using our method, the default flow, labeled *simple inference*, first trains the network using well-known BNN training methods [18], then folds BN layers, and deploys the network, which is to run inference with the network on ReRAM devices possibly laden with various non-idealities. Alternatively, using our method, we take an already trained model, and run a few dozen iterations using a calibration dataset, updating the mean and std. deviation of BN layers. The updated BN layers are folded in the same way as in the simple inference, and the resulting network is deployed.

The FPT step is not a training step, and is more like inference in that no parameter update happens except for the running mean and std. dev. in BN layers. Without SAFs, the statistics (i.e., mean and std. dev.) of each batch will be very similar to the (final) running statistics. But the existence of SAFs will change the batch statistics, which is averaged over several iterations using EMA to generate a new running statistics (EMA is initialized to the final running statistics of training). For DNNs without BN layers, one can add new BN layers next to convolution/fully-connected layers for the sole purpose of SAF mitigation; BN layers are removed after FPT by BN layer folding [17].

During the FPT step we also need information about faulty ReRAM hardware. The information about SAFs can be provided as a fault map [12], which can be used in offline inference with FPT enabled.

C. Analysis: Whether adding an affine transformation can improve the classification accuracy at all

One might ask how adding a simple operation like BN can help improve accuracy at all. To answer, we model a neuron

[2]We do need a calibration set to get the statistics of input, which however can be unlabeled and small.

output and distortion as random variables and see if adding an affine transformation can result in higher classification accuracy. For generality we assume that the neuron output can have up to two peaks and the distortion can have different distribution parameters depending on the target output.

Let X_0, X_1 be random variables with Gaussian distribution modeling the output of a neuron, representing two peaks (see Figure 2a).

$$p(x_0) = \mathcal{N}(x \mid \mu_{x0}, \sigma_{x0}^2), \quad p(x_1) = \mathcal{N}(x \mid \mu_{x1}, \sigma_{x1}^2) \quad (2)$$

The decision boundary, θ, can be chosen to be the point where the two Gaussian probability density functions (PDFs) meet.

Let us assume that the effect of SAF in the weight parameters can be modeled as Gaussian noise with different parameters depending on the target output.

$$G_0 \sim \mathcal{N}(\mu_{g0}, \sigma_{g0}^2), \qquad G_1 \sim \mathcal{N}(\mu_{g1}, \sigma_{g1}^2) \quad (3)$$

Then $X_0' = X_0 + G_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$ and $X_1' = X_1 + G_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$, where $\mu_i = \mu_{xi} + \mu_{gi}$ and $\sigma_i^2 = \sigma_{xi}^2 + \sigma_{gi}^2$ with $i = 0, 1$.

We add an affine layer modeling BN as $Y_i = aX_i' + b$, where the computation of BN is assumed to be free of SAF. Now we can estimate the error probability with the SAF-induced Gaussian noise. Using the previous decision boundary θ, the probability of error with noise and BN applied is given as:

$$e_0 = P(Y_0 > \theta) \quad \text{when the true output is 0,} \quad (4)$$
$$e_1 = P(Y_1 < \theta) \quad \text{otherwise.} \quad (5)$$

Now our objective is to find a, b that can minimize the total error rate $\mathcal{L} = \phi_0 e_0 + \phi_1 e_1$, where ϕ_0, ϕ_1 are the prior probability of the true output ($\phi_0 + \phi_1 = 1$). We take partial derivatives of $\mathcal{L}$ to find a minimum, which gives us the following equation to find the optimal values of a and b:

$$\frac{\phi_0}{\sigma_0} g\left(\frac{\theta - b}{a\sigma_0} - \frac{\mu_0}{\sigma_0}\right) = \frac{\phi_1}{\sigma_1} g\left(\frac{\theta - b}{a\sigma_1} - \frac{\mu_1}{\sigma_1}\right) \quad (6)$$

where g is the standard Gaussian PDF. For a simple case where $\sigma_0 = \sigma_1 = 1$, (6) becomes $\phi_0 g(x - \mu_0) = \phi_1 g(x - \mu_1)$, where $x = (\theta - b)/a$, meaning that x is the point at which the two Gaussians centered at μ_0 and μ_1 meet. Clearly, x will coincide with θ if the two Gaussians (at μ_0 and μ_1) had the same parameters as in (2), i.e., if there were no noise. With noise, (a, b) minimizing $\mathcal{L}$ can be different from $(1, 0)$.

Though finding a closed-form solution to the above equation is difficult, our analysis shows that an affine transformation such as BN can help reduce classification error in the presence of random noise. Note that we only show optimal $(a, b) \neq (1, 0)$ can exist, but we do not claim our FPT method can find the optimal values; our method is only a heuristic.

IV. Experiments

A. Experimental Setup

To evaluate the effectiveness of our proposed method, we use BNNs [18]. Specifically, we use the BNN models in [18] designed for the MNIST and CIFAR-10 datasets with MLP (Multi-Layer Perceptron) and VGG models with $3\times$ inflation factor [19], respectively.

TABLE III: Comparing various training methods and our FPT for MNIST BNN (unbalanced case, OCR=1, 10 epochs). SI refers to simple inference.

MNIST test accuracy					
FR	SI	FPT	Bias train	BN train	Retrain
10%	97.36%	97.47%	97.81%	97.93%	97.32%
20%	91.85%	97.21%	97.23%	97.35%	96.92%
40%	44.23%	95.07%	94.93%	96.7%	97.19%

CIFAR-10 test accuracy					
FR	SI	FPT	Bias train	BN train	Retrain
10%	78.18%	88.83%	89.25%	90.97%	91.40%
20%	32.46%	86.08%	87.21%	89.29%	90.76%
40%	10.08%	66.30%	68.18%	82.39%	87.75%

Our experimental flow is implemented on PyTorch, and supports weight partitioning and mapping to RCAs as well as our FPT method as described in Figure 3. In addition to SAF, we also consider other non-idealities such as IR drop and device variability. For SAF injection we follow the methodology in [10]. IR drop effect is simulated by a SPICE-equivalent simulator, presented in [20]. Variability is modeled as additive Gaussian noise to RCA resistance as follows: $R' = R(1 + \epsilon)$, where $\epsilon \sim N(0, \sigma^2)$. We vary σ from 0.1 to 0.5. We use the following device parameters: $R_{HRS} = 1$ MΩ, $R_{LRS} = 1$ kΩ, crossbar size = 64×64 .

For the calibration set, we use a randomly selected subset of the training dataset. We vary fault rate (FR) and the ratio between stuck-open faults vs stuck-close faults, or Open-Close-Ratio (OCR). In other words, if FR=10% and OCR=4, we can expect about 8% of ReRAM cells stuck to HRS regardless of target resistance value, and about 2% of cells stuck to LRS. We vary FR from 10% to 40% and use OCR values of 5, 1, 1/5.[3]. Our baseline accuracy is the test accuracy of each BNN model, without any non-ideality. Baseline accuracy is 98.00% for MNIST, 91.27% for CIFAR-10.

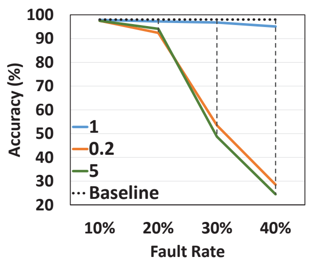
B. Effectiveness of Our FPT Method

Table III compares various partial retraining/full retraining methods with our FPT method and simple inference (as defined in Figure 3). BN training updates backward parameters (β, γ) of BN using back-propagation, but weight parameters of convolution/fully-connected layers are not touched. Bias training updates the bias of each convolution/fully-connected layer only. Up to FR 20%, our FPT result is comparable to that of other training methods; only 3%p drop is observed at FR 40% in MNIST. This result suggests that our FPT method can closely follow (partial) retraining methods.

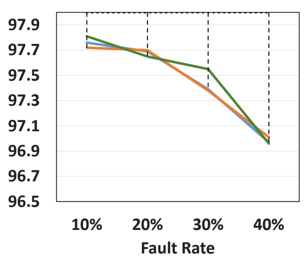
C. Effect of OCR and Weight Realization

In this section we explore several cases of OCR to explore more realistic ReRAM SAF cases, the result of which is summarized in Fig 4a. The OCR values of 5 and 1/5 are chosen based on the literature [5], [9]. The graph shows that

[3]Previous work reports various OCR values including 0.225 [9] and 5.1 [5]



(a) Unbalanced Realization



(b) Balanced Realization

Fig. 4: Comparison between balanced and unbalanced realizations of the weights (MNIST BNN). Different colors mean different OCR values.

TABLE IV: Performance of FPT when using train vs test dataset as the calibration dataset (CIFAR-10 BNN, OCR=1, Unbalanced).

FR	Train dataset	Test dataset
10%	89.74	88.04
20%	87.52	85.36
40%	67.89	67.37

our FPT method is sensitive to OCR value for unbalanced weight realization, with OCR=1 being the best. On the other hand, using balanced realization (see Figure 4b) gives great fault recovery regardless of OCR values. Even we observe that the most skewed OCR values result in quite small accuracy drop of up to 1.04%p when FR=40%.

D. Calibration Dataset

In constructing calibration dataset, we need to know from where to get data and how many images to use. Table IV compares the training vs test dataset as the source of our calibration set. For this experiment, we use the maximum size, i.e., 10000 images, which is the size of the test dataset. Overall, the training set is slightly better than the test set, which may suggest that using independent data (i.e., data that is not used during inference) is okay, meaning it does not cause any performance degradation.

Figure 5 provides an answer to the minimum number of images for the calibration set. We use unbalanced, OCR=1/9 for the worst-case simulation. The graph shows the number of iterations used in FPT by specific batch sizes. The calibration dataset size is the product of the number of iterations and the batch size. Each graph is the average accuracy of 10 independent experiments. The graph clearly shows that the accuracy has stronger correlation with the number of iterations, rather than batch size. Overall, 1024 images (32 iterations, batch size 32) shows only 0.5%p drop from the best case, and thus can be recommended as the minimum calibration set size. In the case where there is not enough images, one guideline is to divide the available data into 32 iterations, which will give a batch size.

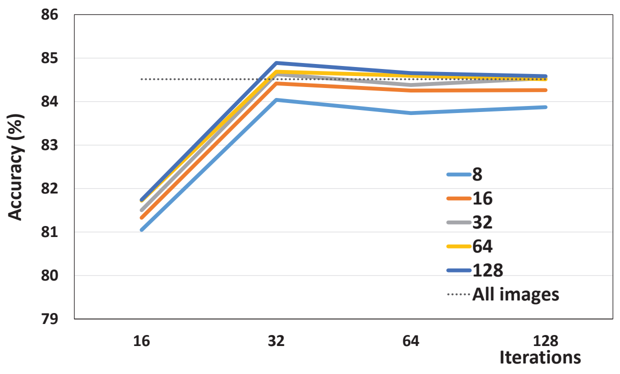


Fig. 5: Performance vs the number of iterations for FPT, with varied batch size (CIFAR-10 BNN, Unbalanced, FR=10%, OCR=1/9). Different colors mean different batch sizes.

TABLE V: Comparison with other mitigation methods. CIFAR-10 test accuracy (OCR = 1/5, Balanced). SI refers to simple inference.

FR	FPT	SI	RF	RP	RF+RP
10%	No	82.32%	87.85%	89.8 %	89.81%
	Yes	88.53%	89.51%	89.37%	89.56%
20%	No	41.21%	57.46%	78.86%	76.38%
	Yes	88.08%	88.37%	88.81%	89.08%

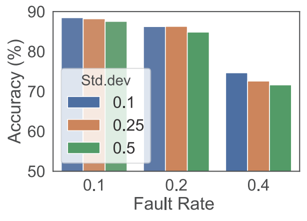
E. Comparison with Previous Methods

We focus on comparison with the remapping methods, and among them, especially row flipping (RF) [13] and row permutation (RP) [8], since they can be applied to any network with relatively low cost (especially compared with the correction methods). Table V summarizes the result of applying various combinations of RF and RP, with and without our FPT method. At 10% fault rate (FR), all mitigation methods tend to show quite high accuracy recovery. However, at 20% FR, the performance of the previous methods starts to deteriorate very badly, that is, unless the FPT method is used together. While FPT alone gives a similar result as applying RF and RP together with FPT (only 1%p difference), between the two, RF may be more cost-effective to apply together due to the differences in the hardware cost and implementation complexity of RF and RP.

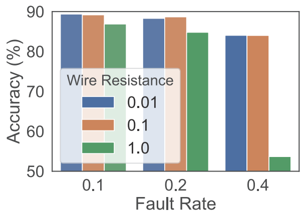
F. Effect of Other RCA non-idealities

ReRAM has many other non-idealities such as variability and IR drop [21]. Variability is considered as one of the major problems for ReRAM especially when compared to other emerging technologies such as STT-RAM. Variability problem may occur during either device programming, reading or both. We have tested the effect of FPT in the presence of both SAF and variability in binary ReRAM. Figure 6a shows the result, where variability is modeled as Gaussian noise to ReRAM resistance values.

The graph shows that accuracy drop steadily increases as variability increases. The graph also suggests that the impact



(a) Variability result



(b) IR drop result

Fig. 6: Non-idealities experiment on CIFAR-10 BNN with OCR = 1/5, Balanced Crossbar.

of variability depends on the fault rate; the higher the fault rate, the greater the impact of variability, which is not surprising. Between device variability and SAF, our result indicates that SAF may be more critical. For instance, while variability of $\sigma = 0.5$ is quite manageable up to FR=20% (<10%p drop), FR=40% makes RCAs quite unacceptable for DNN accelerators regardless of variability.

In the case of IR drop (see Figure 6), accuracy drop is generally smaller than in the variability case, though it has a strong dependence on r_w. Again, our FPT method shows very high resilience, and even up to 40%, we observe only about 6%p drop from the baseline accuracy when wire resistance is low ($r_w = 0.1$). But in the case of high wire resistance ($r_w = 1$), FPT shows about 5% accuracy drop up to FR=20%, but then a very large drop at FR=40%, which cannot be tolerated without retraining.

V. Conclusion

In this paper we presented a novel method to mitigate the effect of permanent faults in RCAs. Our method does not require additional hardware or large datasets, but only updates mean and variance of the batch normalization layers using a tiny fraction of unlabeled data. Batch normalization layers are folded back into preceding layers, thus incurring no additional computation either. In addition our method is orthogonal to remapping techniques as demonstrated by our experimental results, and shows a similar level of fault recovery as back-propagation based training methods, up to about 20% fault rate. Our FPT method also demonstrates good harmony with techniques for other non-idealites, including variability and IR drop. Therefore, FPT is a suitable method to solve the SAF issue in ReRAM device for DNN accelerators.

From our experimental results, it seems that FPT can also help mitigate the effect of other non-idealities such as variability and IR drop, but quantifying its effectiveness requires a more careful study and evaluation. We also plan to apply FPT to larger networks and datasets. Finally, analogue or continuous-valued ReRAM devices can be more useful as they provide higher integration density. Extending our technique to such cases is left for future work.

References

[1] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[2] C. Li *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

[3] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 476–488.

[4] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Fault tolerance in neuromorphic computing systems," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, Jan. 2019.

[5] 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*, vol. 64, no. 1, pp. 180–190, 2014.

[6] G. Charan *et al.*, "Accurate inference with inaccurate rram devices: A joint algorithm-design solution," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 6, no. 1, pp. 27–35, 2020.

[7] L. Xia *et al.*, "Stuck-at fault tolerance in RRAM computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, Mar. 2018.

[8] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, Mar. 2017.

[9] C. Liu, M. Hu, J. P. Strachan, and H. H. Li, "Rescuing memristor-based neuromorphic design with high defects," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, Jun. 2017.

[10] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, "Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, Jun. 2019.

[11] F. Zhang and M. Hu, "Defects mitigation in resistive crossbars for analog vector matrix multiplication," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 187–192.

[12] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, Jun. 2017.

[13] B. Zhang, N. Uysal, D. Fan, and R. Ewetz, "Handling stuck-at-faults in memristor crossbar arrays using matrix transformations," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, Jan. 2019.

[14] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[15] C.-C. Chang *et al.*, "Mitigating asymmetric nonlinear weight update effects in hardware neural network based on analog resistive synapse," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 116–124, 2017.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, 2015, p. 448–456.

[17] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2017.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[20] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and analysis of passive switching crossbar arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 270–282, 2017.

[21] S. Lee, M. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, "Learning to predict IR drop with effective training for ReRAM-based neural network hardware," in *DAC*, Jul. 2020, pp. 1–6.