Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping

Zhezhi He†, Jie Lin†, Rickard Ewetz, Jiann-Shiun Yuan and Deliang Fan
Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
† These authors contributed equally
Elliot.He@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

In this work, we investigate various non-ideal effects (Stuck-At-Fault (SAF), IR-drop, thermal noise, shot noise, and random telegraph noise) of ReRAM crossbar when employing it as a dot-product engine for deep neural network (DNN) acceleration. In order to examine the impacts of those non-ideal effects, we first develop a comprehensive framework called **PytorX** based on main-stream DNN pytorch framework. PytorX could perform end-to-end training, mapping, and evaluation for crossbar-based neural network accelerator, considering all above discussed non-ideal effects of ReRAM crossbar together. Experiments based on PytorX show that directly mapping the trained large scale DNN into crossbar without considering these non-ideal effects could lead to a complete system malfunction (i.e., equal to random guess) when the neural network goes deeper and wider. In particular, to address SAF side effects, we propose a digital SAF error correction algorithm to compensate for crossbar output errors, which only needs one-time profiling to achieve almost no system accuracy degradation. Then, to overcome IR drop effects, we propose a Noise Injection Adaption (NIA) methodology by incorporating statistics of current shift caused by IR drop in each crossbar as stochastic noise to DNN training algorithm, which could efficiently regularize DNN model to make it intrinsically adaptive to non-ideal ReRAM crossbar. It is a one-time training method without the request of retraining for every specific crossbar. Optimizing system operating frequency could easily take care of rest non-ideal effects. Various experiments on different DNNs using image recognition application are conducted to show the efficacy of our proposed methodology.

1 INTRODUCTION

Resistive crossbar memory, as one the most popular memory array structure, has drawn great research interests owing to its high memory accessing bandwidth and *in-situ* computing ability. More importantly, its current-mode weighted summation operation intrinsically matches the dominant Multiplication-and-Accumulation (MAC) in the artificial neural network, making it one of the most promising candidates as the basic computing unit for neural network accelerator design [5, 7]. Resistive Random-Access-Memory (ReRAM) is a two terminal device with programmable resistance, which is taken as the target device in this work. However, many non-ideal effects, such as wire resistance, Stuck-At-Fault (SAF), thermal noise, shot noise, random telegraph noise, etc. [9], are hampering the progress of real hardware implementation of large-scale deep neural network (DNN) on ReRAM crossbar-based accelerator. Many recent works have investigated such issue with either hardware or software solutions, which are summarized as follows.

Hardware approach: In [18], Xu et al. investigate the impact of different resistance allocation schemes, programming strategies, peripheral designs, and material selections in terms of the area, latency, power, and reliability of ReRAM crossbar. The IR-drop caused by the wire resistance of the crossbar is modeled by Liu et al. in [13].

Software approach: Other recent works [2, 3, 9] have adopted different methods that all require retraining the weights of target DNN to be mapped in a crossbar array w.r.t various non-ideal effects for the specific device. However, the main drawback of such method is that the compute-intensive retraining is required for each specific crossbar accelerator, which is definitely not cost-efficient and not practical for future DNN deployment into different crossbar accelerators in the user end. Beyond that, typical anti-nonideal effect retraining considers and adapts the deterministic non-ideal effect (e.g., SAF defects). While, incorporation of many other non-ideal effects (e.g., thermal noise) with stochastic behavior or IR-drop still remain unsolved. Therefore, rather than retraining the DNN to get reconstructed weights every time for a new crossbar accelerator, we propose different techniques for each type of non-ideal effects, which only need one-time optimization for target DNN.

Our main contributions in this work can be enumerated as:

- We first develop a comprehensive crossbar-based simulation framework for DNN training, mapping, and evaluation, called *pytorX*[1]. PytorX is built on top of pytorch, which is one of the most popular python based deep learning framework. Different ReRAM crossbar non-ideal effects, including IR-drop (i.e., wire resistance), Stuck-At-Fault (SAF) defects, thermal noise, shot noise and Random Telegraph Noise (RTN), are modeled as deterministic or stochastic noise to accurately evaluate the performance of DNN on ReRAM crossbar. Based on PytorX, we show that SAF and IR-drop become the main accuracy degradation sources when DNN model goes deeper and wider. While other non-ideal effects have limited effects on accuracy through optimizing the operating frequency.

[1]Code is released at https://github.com/elliothe/PytorX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317870

- To overcome SAF effects on DNN crossbar, we propose a new digital SAF Error Correction method which only needs one-time profiling of the target crossbar without compute-intensive network retraining. It achieves almost no accuracy degradation in ResNet20 for CIFAR10 dataset.
- To overcome the large IR drop in ReRAM crossbar, we propose a Noise Injection Adaption (NIA) methodology by modeling the statistics of current drift caused by wire resistance as a stochastic noise source, then incorporate the modeled noise source into DNN training. Such a method could eventually regularize DNN model to make it intrinsically adaptive to varying IR-drop. Note that, such noise injection is added to the initial DNN training for crossbar mapping and no re-training is needed for each particular accelerator.
- As for thermal, shot and random telegraph noise, we show that crossbar system accuracy is less sensitive when optimizing the operating frequency. Various experiments on different DNNs using image recognition application are conducted to demonstrate the efficacy of our proposed comprehensive methodology.

2 NON-IDEAL EFFECTS AND MODELING

Generally, the non-ideal effects of ReRAM crossbar can be divided into two categories and modeled as deterministic noise n_{d} and stochastic noise n_{s}. In this work, we will consider Stuck-At-Fault (SAF) as the deterministic non-ideal effect, while the stochastic non-ideal effects include effects of crossbar wire resistance, thermal noise, shot noise and Random Telegraph Noise (RTN).

2.1 Deterministic Noise Modeling

2.1.1 device defects. There are two kinds of Stuck-At-Fault (SAF) defects, which are Stuck-At-zero (SA0) and Stuck-At-one (SA1) respectively [17, 19]. The SA0 defect is normally caused by short defects, which makes parts of the ReRAM cells always at high conductance state (i.e., $G_{\max}$). The SA1 defect is resulting from the cell permanent damage and broken selector or wire, which makes parts of the ReRAM cells stuck at low conductance state (i.e., $G_{\min}$). The ReRAM fabrication results in [4] show 1.75% and 9.04% defect rate for SA0 and SA1 respectively. The SA0/SA1 defect rates around the 1.75%/9.04% are taken as the region of interest in this work.

2.2 Stochastic Noise Modeling

2.2.1 wire resistance. In this work, the reason we consider the wire resistance as a stochastic noise term is that the output current degradation of crossbar highly relies on both input voltage and corresponding ReRAM conductance. Modeling it as a deterministic term is extremely computing expensive and not feasible for large scale DNN mapping. For example, in [13], Liu et al. employ a gradient search method to compensate for the weight matrix for IR-drop. However, the process needs to be repeated whenever the inputs of crossbar changes, which requires massive computing resources and training time. In this work, we propose to leverage the statistics of current drift caused by wire resistance, then model it as a stochastic noise source. PytorX will take such noise source into DNN training, which improves the robustness of DNN against the IR-drop. The implementation details are given in in section 4.3.

2.2.2 Thermal Noise and Shot Noise. Thermal noise is typically caused by thermal agitation of the electrical carriers, while the shot noise is induced by the random arrival of the carriers [10]. Both of them can be modeled with random current source following zero-mean Gaussian distribution. Thus we merge them as a single current source presented in parallel with ReRAM and the Root-Mean-Square (RMS) current, which can be expressed as:

$$i_{\mathrm{rms}} = \sqrt{Gf(4K_{\mathrm{B}}T + 2qV)} \tag{1}$$

where G is the ReRAM conductance, f is the operating frequency of the crossbar, K_{B} is Boltzmann constant, T is the temperature in Kelvin, q is electron charge, and V is the voltage drop across the ReRAM two terminals. Furthermore, we also include the programming variation of ReRAM in our noise model. We consider the programming variation follows Gaussian distribution $\mathcal{N}(0, \sigma^2)$, where $\sigma = \Delta G/3$ and ΔG is the ReRAM conductance resolution. Then, the equivalent standard deviation of ReRAM conductance can be described as:

$$g_{\mathrm{rms}} = \frac{i_{\mathrm{rms}}}{V} = \sqrt{\frac{Gf(4K_{\mathrm{B}}T + 2qV)}{V^2} + \left(\frac{\Delta G}{3}\right)^2} \tag{2}$$

2.2.3 Random Telegraph Noise. Random Telegraph Noise (RTN) constitutes a major cause of the errors in ReRAM by temporarily and randomly reducing the resistance [15]. In this work, we adopt the RTN model proposed by Ielmini et al. in [8], where the average ReRAM resistance change (R_{rtn}) strongly depends on the current resistance state of ReRAM (R). The relation can be described as $R_{\mathrm{rtn}}/R = aR + b$, where a and b are fitting parameters. By converting the resistance into conductance, we can obtain $G_{\mathrm{rtn}}/G = \frac{bG+a}{G-(bG+a)}$. With $a = 1.662e-7$ and $b = 0.0015$ reported by [8], we model the RTN as a Poisson process [15]. Thus, the actual ReRAM conductance value under RTN (G_{act}) can be mathematically written as:

$$G_{\mathrm{act}} = \begin{cases} G + G_{\mathrm{rtn}} & \tau < 0.5 \\ G & otherwise \end{cases} \tag{3}$$

where τ is a random number that uniformly distributed between (0,1) (data from [15]).

3 CROSSBAR BASED NN-ACCELERATOR

In this section, we first introduce the single ReRAM crossbar design as the dot-product engine. Then, the network partition method adopted for mapping the large scale DNN across multiple arrays is introduced in the following subsection. Note that, we assume there exists a digital arithmetic unit as the co-processor within the crossbar-based accelerator, to perform signal scaling, addition with bias and other computations like Batch-Normalization layers.

3.1 Single array as dot-product engine

The primary computation of convolution and fully connected layer in deep convolution neural network is the weighted summation with bias offset, which can be rewritten as the element-wise dot-product format:

$$y = \boldsymbol{w}^T \cdot \boldsymbol{x} + b = \begin{bmatrix} \boldsymbol{w}^T \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x} & b \end{bmatrix} \tag{4}$$

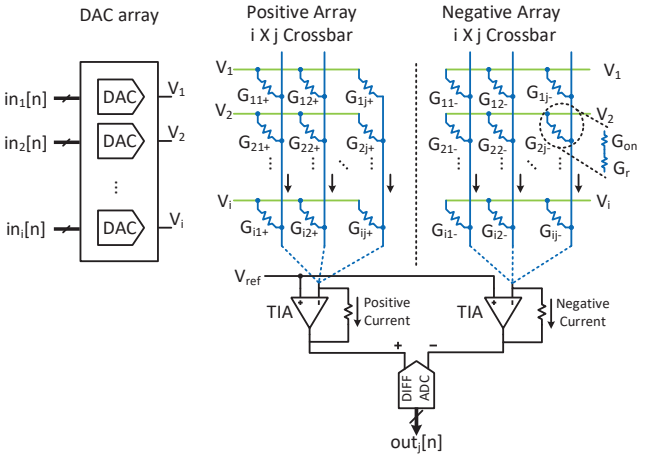

Figure 1: Hardware implementation of single $M \times M$ ReRAM crossbar array pair (positive and negative array) as analog dot-product engine. ReRAM selector is modeled as $G_{\mathbf{on}}$ cascaded with ReRAM conductance $G_{\mathbf{r}}$.

where $\boldsymbol{w}$ and $\boldsymbol{x}$ are vectorized weights and inputs respectively, and b is the bias term. For hardware mapping purpose, both weight ($w \in \mathbb{Z} \cdot \Delta x$) and input ($x \in \mathbb{Z} \cdot \Delta w$) are quantized into fixed-point representation, where Δx and Δw are the fixed-point quantization step size. In this work, we choose 8-bit for both the inputs and the weights, and we emit the bias term where the computation is performed by the digital co-processor. Thus, the above equation can be rewritten as:

$$y = (\Delta w \cdot \Delta x) \sum_i \hat{w}_i \cdot \hat{x}_i \tag{5}$$

The ReRAM crossbar is taken as the analog dot-product engine to accelerate the computation of eq. (5) as shown in fig. 1. Owing to the weights can be either positive or negative, each weight is represented by two ReRAM cells (i.e., $G^+_{i,j}$ and $G^-_{i,j}$) allocated in a positive and a negative array. As shown in fig. 1, the input x_i is encoded as binary bit-strings $\text{in}_i[n]$ for crossbar input, where each one has 8-bit digits ($n = 8$) in 2's complement format. Such binary encoded inputs are converted by the DACs, where the output voltage of the DAC on i-th row can be expressed as:

$$V_{\text{DAC},i} = V_{\text{ref}} + \Delta V_{\text{DAC}} \cdot \hat{x}_i \tag{6}$$

where $V_{\text{ref}} = V_{\text{DD}}/2$ is the reference voltage, and ΔV_{DAC} is the minimum voltage stage of the DAC. Therefore, the current forward into the differential ADC for j-th column pair (i.e., two corresponding columns in the positive and the negative array) can be described as:

$$I_{\text{ADC},j} = \sum_{i=1}^{M} \left((V_{\text{DAC},i} - V_{\text{ref}}) \cdot (G^+_{i,j} - G^-_{i,j}) \right) \tag{7}$$

For ReRAM crossbar programming, there exist two mapping schemes which are equal-ΔR and equal-ΔG respective. Hereby we adopt the more straight-forward equal-ΔG mapping scheme since the equal-ΔR introduces an undesired device-oriented non-uniform quantization.

In order to properly mapping the eq. (5) onto the eq. (7), we rewrite the function as follows:

$$G^+_{i,j} - G^-_{i,j} = \Delta G \cdot \hat{w} \tag{8}$$

Algorithm 1 DNN quantization training scheme.

Require: Layer-wise quantization step of input Δx, weight Δw, and ADC ΔI_{ADC}. Resolution of input, weight, ADC are in $N_{\text{B}}^{\text{dac}}$, $N_{\text{B}}^{\text{rram}}$, and $N_{\text{B}}^{\text{adc}}$ bits. Number of training batches N_{train}.
Ensure: Minimize the loss and obtain fixed Δx, Δw and ΔI_{ADC}.
{1. Training phase of l-th layer in epoch t}
Initialization: $S_{\text{x}} \leftarrow 0$; $S_{\text{I}} \leftarrow S_{\text{I}} + \Delta I_{\text{ADC}}$
for $i := 1$ to N_{train} **do**
  Update $\boldsymbol{w}_l$ through back-propagation
  $\Delta w_l \leftarrow \max(|\boldsymbol{w}_l|)/2^{N_{\text{B}}^{\text{rram}}}$
  $\Delta x_l \leftarrow \max(|\boldsymbol{x}_l|)/2^{(N_{\text{B}}^{\text{dac}}-1)}$
  Compute $\boldsymbol{I}_{\text{ADC}}$ ▷ eq. (10)
  $\Delta I_{\text{ADC}} \leftarrow \max(|\boldsymbol{I}_{\text{ADC}}|)/2^{(N_{\text{B}}^{\text{adc}}-1)}$
  $S_{\text{x}} \leftarrow S_{\text{x}} + \Delta x_l$; $S_{\text{I}} \leftarrow S_{\text{I}} + \Delta I_{\text{ADC}}$
end for
{2. Test phase}
$\Delta x_l \leftarrow S_x/N_{\text{train}}$; $\Delta I_{\text{ADC}} \leftarrow S_{\text{I}}/N_{\text{train}}$

$$G^+,\ G^- = \begin{cases} \Delta G \cdot \hat{w} + G_{\min}, & G_{\min} \quad \text{if } \hat{w} \geq 0 \\ G_{\min}, & \Delta G \cdot |\hat{w}| + G_{\min} \quad \text{if } \hat{w} < 0 \end{cases} \tag{9}$$

where ΔG is the conductance step size of the programmable ReRAM cell. Note that here the conductance G^+/G^- is the series combination of the on-conductance of the selector G_{on} and the ReRAM programmed value $G_{i,j}$. Thus, eq. (7) can be reformatted as:

$$I_{\text{ADC},j} = (\Delta V_{\text{DAC}} \cdot \Delta G) \sum_{i=1}^{M} \hat{x} \cdot \hat{w} \tag{10}$$

Then, with the ADC at the output-end for performing current sensing and quantization function[2], where the LSB current of ADC is configured as $\Delta I_{\text{ADC}} = k \cdot \Delta V_{\text{DAC}} \cdot \Delta G$, the final output is:

$$\hat{y} = \text{round}(\frac{I_{\text{ADC},j}}{\Delta I_{\text{ADC}}}) = \text{round}(\frac{1}{k} \sum_{i=1}^{M} \hat{x} \cdot \hat{w}) \tag{11}$$

In this work, we consider ΔV_{DAC} and ΔG are fixed by hardware, while ΔI_{ADC} is configurable through tuning the reference current source of ADC. To prevent the accuracy degradation caused by the quantization error, we also propose a crossbar-aware network quantization method as described in algorithm 1. Rather than minimizing the quantization error w.r.t each input, our method optimizes the layer-wise Δw, Δx and k, then return them as the fixed value. Thus, the computation workload on digital co-processor is minimized.

3.2 Network Partition on Matrix Arrays

Nowadays, the parametric layers of the state-of-the-art DNN normally contain a large number of weights which exceeds the size of a single crossbar. Thus, each DNN layer may require multiple crossbar arrays for weight accommodation. A network partition technique is necessary here for efficient weight mapping and network computation. For simplicity, we adopt the naive network partition method similar as introduced in [16], which converts the weight tensor of each convolution/fully-connected layer into two

[2]In this work, we use the eq. (11) for simplicity. For a more commonly used ADC quantization, $\hat{y} = \text{round}(I_{ADC,j}/\Delta I_{\text{ADC}} - 0.5) + 0.5$ is also an option.

Table 1: Parameters of ReRAM crossbar system.

Symbol	Description	Value
f	Operating frequency	0.01~1 GHz
V_{dd}/V_{ref}	Supply/reference voltage	3.3/1.67 V
G_{min}/G_{max}	ReRAM min/max conductance	333/0.33 μS
ΔG	Conductance step size	2.601 μS
$N_B^{\{adc,dac\}}$	Resolution of ADC/DAC	8/8 bit
T	Temperature	300~350 K
C_{size}	Crossbar Dimension	32, 64, 128

4-D Matrix Arrays with shape $\{N_{row},N_{col},C_{size},C_{size}\}$ as G^+ and G^- respectively, where $N_{row}\times N_{col}$ is the number of crossbar arrays used for one layer.

4 END-TO-END NETWORK ADAPTION

In this section, we will explicitly discuss the impacts of different non-ideal effects on the ReRAM crossbar based DNN acceleration, and the effectiveness of proposed countermeasures for preventing accuracy degradation.

4.1 Simulation Framework and Configurations

We divide the experimental setup into hardware and software parts for the clarification of the tool-chain and models used in this work.
Hardware: The state-of-the-art fabrication result has shown the 6-bit [20] to 7-bit [1] ReRAM programmable resistance level. In this work, we choose moderate resolutions for ADC, DAC and ReRAM (i.e., 8-8-7 bit), other chosen parameters are tabulated in table 1. Partial of the parameters chosen for the non-ideal effects setup are provided in section 2. The ReRAM crossbar-based neural network accelerator is operating as introduced in section 3.
Software: In order to perform a comprehensive investigation on ReRAM crossbar based neural network accelerator under various non-ideal effects, we build a comprehensive simulation framework called *PytorX*, which is implemented in Python and built upon PyTorch. It models the crossbar computation based on the discussion and equations in section 3, including signal/unit conversion (i.e., DAC, ADC, and weight mapping), weight partition on multiple arrays, and etc. PytorX is a GPU-favored framework and fully relies on the tensor operation, thanks to the rich APIs provided by the Pytorch. In order to demonstrate the effectiveness of our proposed method, we take the classical object recognition task as an example. A variety of DNN architectures (LeNet-5 variant [3] and ResNet-20 [6]) on different scale datasets (MNIST [12] and CIFAR-10 [11]) are studied in the following sections. Hyper-parameters configuration and image argumentation method adopted in this work are following their original works, which will not be listed here.

4.2 Error Correction for SAF

Many previous studies [3, 17] only discuss the effect of SAF on NN inference, using a simple 2-layers fully-connected network on MNIST, where the results show that SAF indeed affects the accuracy but with a relatively low degradation. However, what we found in our experiments is that such accuracy degradation drastically increases with wider and deeper network architecture on a larger dataset. fig. 2 depicts the test accuracy on MNIST and CIFAR-10 dataset using LeNet-5 and ResNet-20 respectively, where the blue shadow indicates the region of interest (i.e., experimentally reported SAF rate). Note that, the SAF rate we used is 1.75% for SA0 and 9% for SA1, respectively. As shown in fig. 2, when directly mapping the network into the crossbar system without any error correction techniques, 1% SA0 defect rate can lead to a complete system malfunction on CIFAR-10 dataset with ResNet-20.

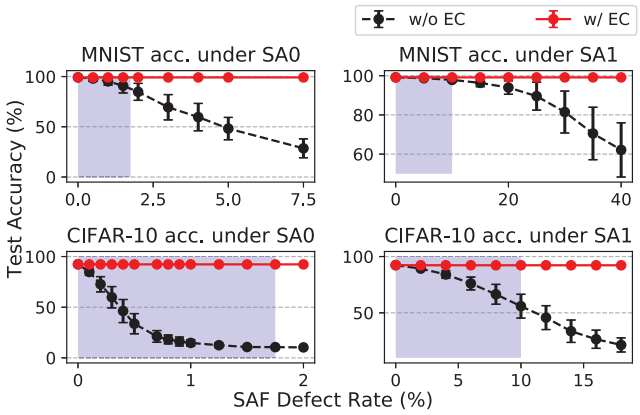

Figure 2: Test accuracy versus SA0/SA1 rate, w/o or w/ Error Correction (EC). (Top) LeNet-5 on MNIST dataset; (Bottom) ResNet-20 on CIFAR-10 dataset. Error-bar denotes mean±std with 100 trials, and crossbar size is 64. Regions with blue shadow are regions of interest.

As the countermeasure to both SA0 and SA1, in this work, we propose a digital SAF Error Correction (EC) solution to compensate the computation error with the digital co-processor. Such method can be generally described into steps: **1)** Profiling the current SAF status of ReRAM crossbar system, then indexing the ReRAM cells with SA0/SA1 defects using binary tensor $\boldsymbol{i}^{sa0}/\boldsymbol{i}^{sa1}$; **2)** Calculating the crossbar output difference $\boldsymbol{y}_{ec}$ between the non-ideal output caused by SAF and the ideal output, where the details are elaborated in algorithm 2; **3)** Adding the yield $\boldsymbol{y}_{ec}$ at the crossbar output utilizing the digital units. Normally, the addition operation in the last step can integrate with the following Batch-Norm layer (i.e., Affine function). For CIFAR-10 test accuracy reported in fig. 2, the SAF-free accuracy is 92.39%. With our error correction method, the worst-case accuracy we obtain is $92.23 \pm 0.08\%$, where the accuracy gap is negligible. Moreover, the binary SAF indexing tensor can use the sparse encoding for efficient storage and computation. Note that, our proposed EC is a local correction method where the DNN retraining is not involved.

4.3 Noise Injection Adaption for IR-drop

The IR-drop caused by wire resistance is another dominant factor which may cause the system malfunction. With a specific wire technology, the IR-drop is not only determined by the crossbar dimension but strongly correlated to both input voltage magnitudes and conductance distribution of ReRAM cells. To visualize the impact of crossbar dimension, we plot the distribution of normalized voltage drop (V'/V) across ReRAM cells in fig. 3, where V' is the voltage drop considering IR-drop, while V is the ideal case without

[3] https://github.com/pytorch/examples/blob/master/mnist/main.py

Algorithm 2 Error Correction (EC) for Stuck-At-Fault (SA0/SA1).

Require: input voltage tensor $\boldsymbol{V}$, ideal weight conductance tensor w/o SAF $\boldsymbol{G}_+$ and $\boldsymbol{G}_-$, non-ideal weight conductance tensor w/ SAF $\boldsymbol{G}_+^{\text{saf}}$ and $\boldsymbol{G}_-^{\text{saf}}$. SA0 indexing tensor $\boldsymbol{i}_+^{\text{sa0}}$ and $\boldsymbol{i}_-^{\text{sa0}}$, SA1 indexing tenor $\boldsymbol{i}_+^{\text{sa1}}$ and $\boldsymbol{i}_-^{\text{sa1}}$. SA0 and SA1 leads to $G_{\max}$ and $G_{\min}$ respectively. ADC current-to-digital conversion $f_{\text{adc}}(\cdot)$.

Ensure: for the given input $\boldsymbol{V}$, the output of ReRAM crossbar is corrected using error compensation w.r.t the profiled SA0 and SA1 information.

$\boldsymbol{y} \leftarrow f_{\text{adc}}(V \times \boldsymbol{G}_+^{\text{saf}} - V \times \boldsymbol{G}_-^{\text{saf}})$ ▷ Crossbar output

$\boldsymbol{G}_+^{\text{diff}} \leftarrow (\boldsymbol{G}_+ - G_{\max}) \cdot \boldsymbol{i}_+^{\text{sa0}} + (\boldsymbol{G}_+ - G_{\min}) \cdot \boldsymbol{i}_+^{\text{sa1}}$

$\boldsymbol{G}_-^{\text{diff}} \leftarrow (\boldsymbol{G}_- - G_{\max}) \cdot \boldsymbol{i}_-^{\text{sa0}} + (\boldsymbol{G}_- - G_{\min}) \cdot \boldsymbol{i}_-^{\text{sa1}}$

$\boldsymbol{y}_{\text{ec}} \leftarrow f_{\text{adc}}(V \times \boldsymbol{G}_+^{\text{diff}} - V \times \boldsymbol{G}_-^{\text{diff}})$ ▷ output diff. w.r.t SAF

return $\boldsymbol{y}_{\text{out}} \leftarrow \boldsymbol{y} + \boldsymbol{y}_{\text{ec}}$ ▷ Digital error compensation

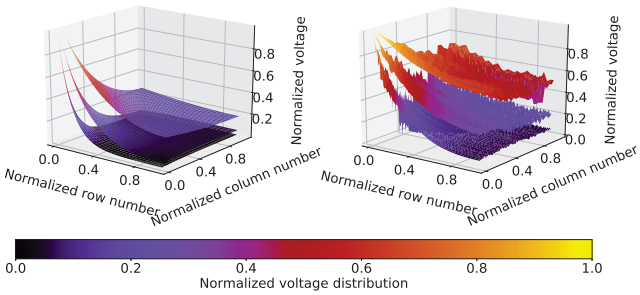

Figure 3: Distribution of Voltage drop of crossbar with different dimensions. Surfaces from top down are 32×32, 64×64 and 128×128 size respectively. (Left) is worst case, (Right) is normal case

IR-drop. fig. 3 indicates that choosing 128×128 as crossbar size is impractical for an accelerator design due to huge IR drop.

In order to study the impact of IR-drop on DNN inference accuracy, we integrate a dynamic crossbar solver into our PytorX simulation framework, based on the simplified Modified Nodal Analysis [14]. The test accuracy on MNIST and CIFAR-10 with IR-drop considered is reported in table 2, where directly mapping the network on 64×64 crossbar results in more than 65% accuracy degradation. For mitigating such significant accuracy loss, we initially attempt to optimize DNN training considering such IR-drop effects, through counting the real-time current shift of each crossbar during the weight update and make it adaptive to existing IR drop effect. However, the biggest problem is that the existing crossbar IR drop solver requires massive computation resources to get real-time shifted current based on current inputs, which also grows exponentially with larger crossbar size. In our experiment, a simple LeNet-5 training takes all of the 64GB main memory and costs days to converge in a 4 Nvidia TitanX GPU workstation, which is too computing-intensive if considering such accurate IR-drop effects. Note that, it will become even worse if the network becomes deeper and wider.

Thus, we propose a Noise Injection Adaption (NIA) method which can statistically approximate the effect of IR-drop and incorporate it into network training, for balancing computation resource and the accuracy of the trained model. Taking LetNet-5 on MNIST as an example, our NIA can be divided into three steps: (1) we first train a LetNet-5 without considering IR-drop during training. Then, with the trained LetNet-5 mapped on the crossbar matrix arrays in our PytorX, we can collect the crossbar output current shift with or without IR-drop for all of the testing data; (2) we approximate the effect of IR-drop as a Gaussian noise source at each crossbar output end, with crossbar-wise mean and standard deviation extracted from the collected statistics; (3) we train the network again with such approximated additive IR drop noise applied at each output of crossbar arrays. The only overhead of our NIA method is a network optimization step which has similar timing cost as a typical DNN training. After the network optimization with NIA, accuracy degradation is significantly reduced as reported in table 2.

Table 2: Test accuracy on MNIST with various crossbar dimensions and our proposed methods

Dataset (crossbar size)	MINIST (32×32)	MNIST (64×64)
Software	99.04%	99.1%
Direct Mapping with EC	96.2%	**32%**
Optimized with NIA	99.1%	**98.3%**
Optimized with NIA + EC + frequency tuning	99.0%	98.1%

4.4 System configuration for Other stochastic non-ideal effects

As discussed in section 2.2, we categorize the thermal noise, shot noise and random telegraph noise as ReRAM stochastic noise to study in this work, where the actual ReRAM conductance with additive ensemble stochastic noise can be modeled as:

$$G_{\text{act}} = G + G_{\text{g}} + G_{\text{p}} \tag{12}$$

where $G_{\text{g}} \sim \mathcal{N}(0, g_{\text{rms}}^2)$ is the Gaussian term as in eq. (2), and Poisson term is $G_{\text{p}} = G_{\text{rtn}}$ under the Poisson process as discussed in eq. (3). It is noteworthy that the Gaussian term is highly correlated with both the ReRAM programmed conductance and crossbar input voltages, which makes the conductance variation of ReRAM cells differs w.r.t each input, which is considered in the simulation. We first try to apply the NIA technique to retrain the neural network for adapting those stochastic terms described in eq. (12). However, our simulation turns out injecting a Gaussian noise $\mathcal{N}(0, \Delta G)$ to the weight equivalent conductance G does not reduce the accuracy degradation. On the contrary, it reduces the inference accuracy. Our explanation to such observation is that injecting Gaussian noise on the weight performs L-2 norm regularization, which leads to entire ReRAM arrays have smaller conductance. However, according to RTN modeling discussed in section 2.2.3, ReRAM cell with smaller G owns larger RTN variation which leads to incorrect inference results. Therefore, in this subsection, rather than trying to adapt the neural network to the stochastic noise terms in eq. (12), we attempt to mitigate thermal noise, shot noise and RTN side effects simultaneously through optimizing system operating frequency. The examination of inference accuracy of crossbar system under varying operating frequency (f) and temperature (T) is shown in fig. 4. It shows that lowering the operating frequency can effectively reduce the thermal noise and shot noise caused accuracy degradation. Then, the left-over terms of programming variation and RTN

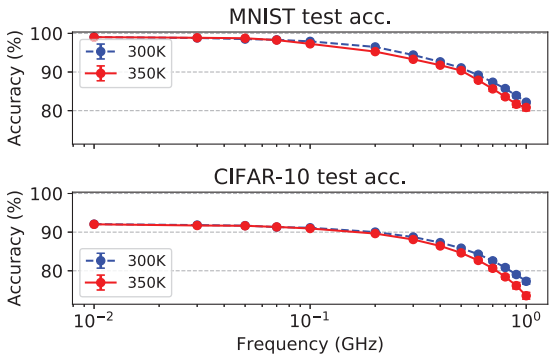

Figure 4: Test accuracy on MNIST and CIFAR-10 dataset versus operating frequency f.

do not show significant impacts on the inference accuracy when f is reduced. We also perform a Monte Carlo simulation with 10000 trails to observe ReRAM conductance variation when $G = G_{\min}$. As shown in fig. 5b, when the f is lowered to 10 MHz, the conductance variation is almost located within the quantization boundary $(-\Delta G, +\Delta G)$. Note that when f is 1 GHz, the signal is buried in the noise floor, and the magnitude and direction of the current through the ReRAM is totally random. As a result, negative conductance can be observed in fig. 5a.

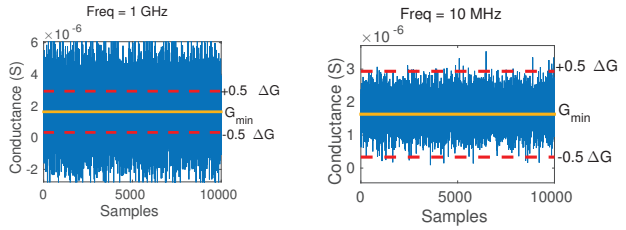

Figure 5: Monte Carlo simulation of the G_{act} under (a) $f = 1GHz$ and (b) $f = 10MHz$, with 10000 trials. The solid line indicate the ideal conductance (G_{min} in this simulation), and dashed line is the quantization boundary.

Moreover, we perform the simulation which takes all the aforementioned non-ideal effects with the proposed NIA, EC and system frequency tuning to minimize the accuracy drop, which is reported in table 2. Note that, the accuracy reported in "Optimized with NIA" does not consider thermal, shot and RTN noise.

5 CONCLUSION

Through PytorX, we are able to perform comprehensive simulation large-scale DNN mapped to ReRAM crossbar-based accelerator accurately. To overcome SAF effects, we proposed a digital SAF error correction algorithm to achieve almost no accuracy degradation in DNN mapping without the need of network retraining. To overcome IR drop effect, we proposed a noise injection adaption method to model IR drop as a stochastic noise source in DNN training to regularize DNN model to make it adaptive to such non-ideal effect. As for thermal, shot and random telegraph noise, we investigated to optimize system frequency to eliminate its degradation in accuracy. Our proposed comprehensive methods together could effectively mitigate the non-ideal effects of ReRAM crossbar and provide great potential for future large scale accurate DNN crossbar based accelerator.

Acknowledgement: This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

REFERENCES

[1] Fabien Alibart et al. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201.
[2] Chakraborty et al. 2017. Technology Aware Training in Memristive Neuromorphic Systems based on non-ideal Synaptic Crossbars. *arXiv preprint arXiv:1711.08889* (2017).
[3] Chen et al. 2017. Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar. In *DATE*. European Design and Automation Association, 19–24.
[4] Ching-Yi Chen et al. 2015. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Trans. Comput.* 64, 1 (2015), 180–190.
[5] Ping Chi et al. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.
[6] Kaiming He et al. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.
[7] Zhezhi He et al. 2017. A tunable magnetic skyrmion neuron cluster for energy efficient artificial neural network. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 350–355.
[8] Daniele Ielmini et al. 2010. Resistance-dependent amplitude of random telegraph-signal noise in resistive switching memories. *Applied Physics Letters* 96, 5 (2010), 053503.
[9] Shubham Jain et al. 2018. Rx-Caffe: Framework for evaluating and training Deep Neural Networks on Resistive Crossbars. *arXiv preprint arXiv:1809.00072* (2018).
[10] LB Kish et al. 2000. Noise in nanotechnology. *Microelectronics Reliability* 40, 11 (2000), 1833–1837.
[11] Alex Krizhevsky et al. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.
[12] Yann LeCun et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[13] Beiye Liu et al. 2014. Reduction and IR-drop Compensations Techniques for Reliable Neuromorphic Computing Systems. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '14)*. IEEE Press, 63–70.
[14] Lawrence T Pillage et al. 1995. *Electronic circuit and system simulation methods*. McGraw-Hill New York.
[15] Puglisi et al. 2015. A complete statistical investigation of RTN in HfO_2-based RRAM in high resistive state. *IEEE Transactions on Electron Devices* 62, 8 (2015), 2606–2613.
[16] Linghao Song et al. 2017. Pipelayer: A pipelined reram-based accelerator for deep learning. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 541–552.
[17] Xia et al. 2018. Stuck-at Fault Tolerance in RRAM Computing Systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 1 (2018).
[18] Cong Xu et al. 2013. Understanding the trade-offs in multi-level cell ReRAM memory design. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 1–6.
[19] Cong Xu et al. 2015. Overcoming the challenges of crossbar resistive memory architectures. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 476–488.
[20] S. Yu et al. 2015. Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect. In *2015 IEDM*. 17.3.1–17.3.4.