Learning to Predict IR Drop with Effective Training for ReRAM-based Neural Network Hardware

Sugil Lee[1,2], Giju Jung[1,2], Mohammed E. Fouda[3], Jongeun Lee[1,2], Ahmed Eltawil[3], and Fadi Kurdahi[3]

[1]School of Electrical and Computer Engineering, UNIST, Ulsan, Korea
[2]Neural Processing Research Center, Seoul National University, Seoul, Korea
[3]Department of Electrical Engineering and Computer Science, University of California-Irvine, CA, USA

***Abstract*—Due to the inevitability of the IR drop problem in passive ReRAM crossbar arrays, finding a software solution that can predict the effect of IR drop without the need of expensive SPICE simulations, is very desirable. In this paper, two simple neural networks are proposed as software solution to predict the effect of IR drop. These networks can be easily integrated in any deep neural network framework to incorporate the IR drop problem during training. As an example, the proposed solution is integrated in BinaryNet framework and the test validation results, done through SPICE simulations, show very high improvement in performance close to the baseline performance, which demonstrates the efficacy of the proposed method. In addition, the proposed solution outperforms the prior work on challenging datasets such as CIFAR10 and SVHN.**

***Index Terms*—ReRAM crossbar array, IR drop, Deep neural network, Binary neural network, 0T1R**

I. Introduction

While ReRAM (Resistive RAM) crossbar arrays (RCAs) are considered as one of the most promising technologies for highly efficient neural network hardware, much of its promise critically depends on the assumption that RCAs can function as a computing unit as well as a storage unit [1]. In an ideal condition, an RCA can do parallel matrix-vector multiplication (MVM) between a *weight* conductance matrix and an *input* voltage vector, with the time complexity of $O(1)$ instead of $O(n^2)$. RCAs can not only provide such an extremely parallel matrix operation, but also eliminate the von Neumann bottleneck since the computation is done right where the weight matrix is stored. For neural network hardware that spends most of its energy doing matrix multiplications, RCA could be an ideal technology, solving both computation and communication problems simultaneously with radically improved efficiency. Furthermore, a passive RCA, one that consists of memristors only without active transistors (i.e., 0T1R), has superior area advantage.

However, if an RCA cannot function as a computing unit, for instance due to its MVM result being unrecoverably distorted, then an RCA would degenerate into a memory, annulling the great promises mentioned above. Such is the threat that can be caused by IR drop when array size is large or wire resistance increases relative to LRS (low-resistance state) resistance. Since the size of RCA arrays is expected to grow, and device-to-wire resistance ratio depends on many factors including material choice, it is important to address the IR drop problem at the system level as much as possible.

In this paper for passive RCAs, we propose a novel method based on statistical machine learning that can learn as well as recall the distortion pattern of MVM computation for any given input, in a fast and reliable manner. Our method enables greatly enhanced training of RCA-based neural networks, significantly increasing the range of usable device-to-wire resistance ratios and RCA sizes. Though in this paper we use BNNs for demonstration due to the maturity of the binary ReRAM technology and the larger array sizes it affords as well as the extreme efficiency from the lack of very costly ADC/DAC circuitry, our method is applicable to multi-bit quantized neural networks as well.[1]

In this paper we make the following contributions.

- We interpret the IR drop problem in passive RCA-based MVM computation as a function identification problem, making it accessible to a number of known techniques such as regression and machine learning.
- We propose two prediction models including a convolutional neural network (CNN) to accurately learn and predict the distortion pattern caused by IR drop.
- We show empirically that our training method is effective in mitigating the effect of IR drop in passive RCA-based neural network hardware for various RCA sizes and wire resistance values.

II. Background and Previous Work

A. IR Drop in ReRAM Crossbar Array

Fig. 1 illustrates a passive RCA of size $n \times m$ performing an MVM operation between a weight matrix W of size $m \times n$ and

[1]BNN is only easy for hardware implementation, but not necessarily easier in terms of training compared to multi-bit neural networks.

This work was supported by Samsung Advanced Institute of Technology, NRF grants funded by MSIT of Korea (No. 2016M3A7B4909668, No. 2017R1D1A1B03033591, No. 2020R1A2C2015066), and Free Innovative Research Fund of UNIST (1.170067.01).

J. Lee is the corresponding author (E-mail: jlee@unist.ac.kr).

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

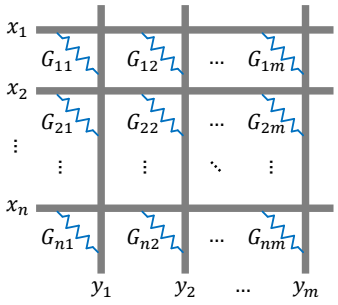


Fig. 1: Performing MVM computation on a passive RCA.

an n-dimensional input vector x, generating an m-dimensional output $y = Wx$, where x is given as voltage, y in current, and W is programmed as conductance after transpose ($G = W^T$).

This works only if the voltage across the two terminals of a ReRAM cell labeled with G_{ij} is exactly x_i. In a crossbar structure it is usually not the case. For instance, if wire has resistance, the input voltage x_i is divided between wire and ReRAM cells, generating less current than $G_{ij}x_i$. Naturally the farther the ReRAM cell is from input/output ports, the higher the IR drop is.

The amount of IR drop depends on ReRAM cells' resistance, which is programmable, as well as wire resistance. Thus the single most important architecture parameter determining the overall severity of IR drop is the LRS-to-wire resistance ratio. But to find out the exact amount of output current, one needs SPICE simulation.

In addition to device faults and variation, ReRAM resistance may drift due to stochastic noise, which in turns affects the amount of IR drop. Therefore all these effects must be considered together, which requires a huge amount of computing resources (on the order of days with a capable workstation), even to evaluate the *inference* accuracy of an MNIST-level neural network.

B. Target Application

Our target application is RCA-based neural network hardware, that is, a hardware accelerator for neural networks where matrix multiplication is performed on RCAs [2]. A large matrix multiplication is divided into smaller ones (via loop tiling), each of which is mapped to an RCA, with necessary input/output routing circuitry added using CMOS technology. All RCAs are dedicated, i.e., completely parallel, since programming ReRAM is costly. Convolution operation can be implemented using matrix multiplication [3].

BNNs often have batch normalization layers [4]. Batch normalization (BN) consists of shift and scaling operations, and help training converge faster. BN is crucial, not just for BNNs but all low-precision networks. The problem with BN is that the computation is a bit complicated, including a division operation. However, for BNNs we can actually eliminate BN layers by modifying the bias values of the preceding layer [5], which is another advantage of BNNs over low-precision multi-bit networks. Note that the bias values in BNNs, whether before or after BN elimination, does not require very high precision, and much less than in multi-bit networks.[2]

C. Related Work

Several ReRAM based neural network architectures have been proposed [2], [6], [7], which are all based on MVM operation on RCA. ISAAC [2], for instance, carefully optimizes the system throughput and cost while supporting multi-bit weight and activation. However, the RCA output, which is analogue, needs to be converted to digital through costly ADC, which is shared and becomes a performance bottleneck. Some ReRAM based neural network architectures [8], [9] support training as well as inference using ReRAM arrays. However all these architectures assume perfect MVM operation, and do not address the training problem in the presence of distorted MVM operation such as when using passive RCAs for superior area advantage.

Consideration of IR drop needs to be done in both inference and training of passive RCA-based neural networks. [10] presents one such framework, with two findings. First, the accuracy drop can be severe: 99% classification accuracy when IR drop is ignored, plummets to mere 32% when IR drop is considered, for LeNet-5 network with MNIST dataset, using 64×64 RCAs. Second, the RCA size is a critical parameter; in the same setting when the RCA size is 32×32, the accuracy drop is only about 3%.

They also propose NIA, which is to model, as Gaussian noise, RCA *output current's shift* via IR drop-enabled inference simulation, and then generate the same Gaussian noise to be added to RCA outputs during training. This way, NIA can avoid costly IR drop simulation (during training) while training a network with IR drop effect taken into account.

There are two problems with this method. First, using an *additive* term to model the effect of IR drop may not be right. Second, the IR drop pattern is inherently 2D, varying across rows and columns of an RCA, which is ignored by focusing on the output of RCA, which is 1D.

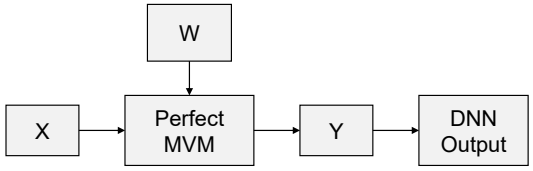
The mask method [11] captures the IR drop pattern as a *2D matrix* that has the same size as the RCA. The mask is *multiplied* to a weight matrix in an element-wise manner before MVM computation. The mask method works because the IR drop pattern has a high spatial correlation, which is captured by mask.

III. Training Under Distorted MVM

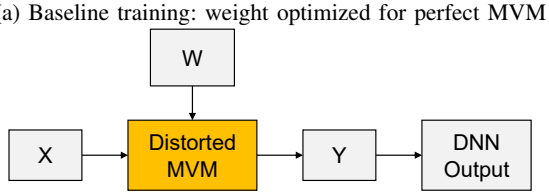
The problem we deal with in this paper is to find the best weight of a DNN in the presence of MVM distortion such as IR drop. In this work we only deal with deterministic nonidealities, though we also evaluate our technique in the presence of stochastic noise (see Section V-E).

[2]Since only the sign of the output activation needs to be determined, and all weight/input activations are binary, the bias only needs to be $\lceil \log_2 n \rceil$-bit, where n is the number of terms added in the computation of output activation.

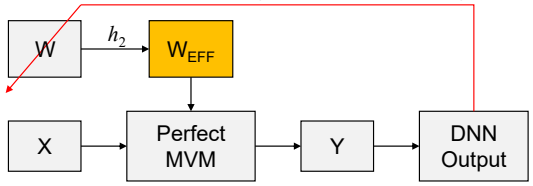
Step 1: $W_1 \leftarrow$ Baseline training
Step 2: Validate(W_1)
Step 3: $W_2 \leftarrow$ Train under distorted MVM
Step 4: Validate(W_2)



(a) Baseline training: weight optimized for perfect MVM



(b) Validation: inference using distorted MVM (e.g., SPICE)



(c) Example of training under distorted MVM

Fig. 2: Our experimental methodology.

A. Methodology

Figure 2 illustrates our experimental methodology. The first step is **baseline training**. The weight obtained from baseline training, if programmed to an accelerator that has perfect MVM computation (e.g., digital CMOS), should give the identical accuracy as in software inference. In this case, $Y = WX$, where Y is MVM output, W weight matrix, and X input activation.

Second, if we apply the weight from the baseline training to an RCA-based accelerator, the distorted MVM will cause poor accuracy. In this case, Y is still a function of W and X but not WX; i.e., $Y = g(W, X) \neq WX$. To find g out, we need simulation such as SPICE, which takes long but is feasible if we do inference only. This step is referred to as **validation**, since if the accuracy in this setup is shown to be high enough, it confirms that the weight has been correctly optimized under MVM distortion.

Third, the validation setup is not useful for training due to two reasons: (i) running IR drop simulation inside a training loop is extremely time-consuming, and (ii) such simulation is not differentiable, thus no guarantee on convergence. Instead our novel approach is to use a *surrogate model* that can predict the outcome of the distorted MVM in a fast and differentiable way, so that we can use the model for training. Once a suitable model for g is found, it is substituted for distorted MVM computation (i.e., SPICE simulation), and we run training again to find a new W under MVM distortion.

Finally, we run validation again to see if the weight from Step 3 gives good accuracy when using the real distorted MVM such as SPICE simulation. Note that there can be quite a difference between the (test) accuracy evaluated with the surrogate model (i.e., accuracy from Step 3), and the validation accuracy obtained in Step 4. This is because the surrogate model may not agree with SPICE simulation for every input combination. The validation accuracy from Step 4 is the main evaluation metric we use in this paper.

B. Linearity Principle

While the number of ways to model distorted MVM computation is unbounded, the following shows three possible ways, according to which we can classify previous methods.

$$Y = g(W, X) \stackrel{?}{=} h_1(WX) \quad (1)$$

$$\stackrel{?}{=} h_2(W)X \quad (2)$$

$$\stackrel{?}{=} W \cdot h_3(X) \quad (3)$$

NIA [10] assumes (1) and further that h_1 is an addition. Mask [11] assumes (2) and that h_2 is an element-wise multiplication. Equation (3) is another possibility, but does not seem very useful in modeling MVM distortion.

We show that (2) is sound for IR drop-induced distortion. To show, we run IR drop simulation [10] using trained weights of an MNIST BNN as W. The network is identical to the one used in Section V except that it has 1024 hidden neurons in each hidden layer. Note that the network's weights are spread across 736 crossbar tiles of 64×64, meaning that there are 736 different weight matrices to use for this experiment. For X we use input activations during MNIST inference (for the first 500 test images). We record Y vectors, which are output current from crossbar arrays obtained via IR drop simulation.

From the collected X, Y pairs, we set up a linear equation, $W_e X = Y$, one per each crossbar, which is underparameterized and can be solved for W_e to minimize the mean squared error, $\mathcal{E} = \text{mse}(W_e X - Y)$. We report the error using the W_e found. If (2) is sound, the error should be very small. Indeed the error turns out to be within rounding error (2.09E-11), confirming that (2) is sound.

The soundness of (2) can be explained as follows. If we limit ourselves to the steady-state behavior during MVM computation, a ReRAM crossbar array can be seen as a resistive network with constant, albeit programmable, resistance values. Then, despite the complex pathways in the crossbar, the output current should be linear to the input voltage, meaning scaling and superposition properties on X should hold, hence (2).

Thus we only need to predict W_e from W. We call the former *effective weight* and the latter *programmed weight*. To recap, the mask method [11] uses the following simple function:

$$W_e = W \circ M \quad (4)$$

where $\circ$ is the Hadamard product (i.e., element-wise multiplication) and M has the same size as W. The intuition behind

TABLE I: BNNs and training parameters

	MNIST	CIFAR10	SVHN
Network type	MLP	CNN	CNN
#Layers	4	9	9
Initial training #epochs	100	500	200
Retraining #epochs	50	200	80
Baseline test accuracy on GPU	98.41%	88.62%	97.18%

this method is that the physical location within an RCA has a dominant effect on the deviation of W_e from W. As we show in Section V, this method has limited accuracy. This is because mask can capture only the first-order effect (i.e. distance from input/output) but ignores second-order effect such as the resistance values of ReRAM cells, which is also important to get correct W_e.

IV. Prediction of Effective Weight

While the problem of predicting W_e from W can be seen as a regression problem, typical regression functions such as linear regression cannot be directly applied due to the high-dimensionality of input/output data in our problem. With a 128×128 RCA, for instance, both input W and output W_e have 16K dimensions, requiring 256 million parameters for linear regression. We suggest two neural network models that can effectively predict W_e as follows.

A. Row-Column Network

Row-column network (RCN) is a nonlinear model, built by stacking multiple neural network layers. IR drop exists in both row and column directions, thus combining row-wise and column-wise models makes sense. We first define two layers, and combine them to create RCN.

Parallel linear layer: We divide the input and output matrices into rows. Each row has its own linear regression parameters, followed by nonlinear activation function. In what follows, we use X and Y to refer to generic input and output matrices (of the same size as W):

$$\mathbf{PL_{row}} : X \to Y \text{ where } Y^{[i]} = f(X^{[i]} \cdot R_i + \vec{b_i}) \quad (5)$$

where $X^{[i]}$ is the i^{th} row of X and f a nonlinear activation function (e.g., *tanh*), R_i a weight matrix of size $m \times m$, and $\vec{b_i}$ an m-dimensional bias vector. In neural network frameworks, this layer can be easily implemented as a hierarchy of primitive layers: a parallel layer and linear layers below it (hence called *parallel linear layer*).

Column-wise parallel linear layer is defined similarly:

$$\mathbf{PL_{col}}(X) = \mathbf{PL_{row}}(X^T)^T \quad (6)$$

Then *row-column network* is defined to capture both row- and column-wise dependency by stacking parallel linear layers of both direction:

$$\mathbf{RCN}: \quad W_e = \mathbf{PL_{col}}(\mathbf{PL_{row}}(W)) \circ U \quad (7)$$

where U is an $n \times m$ parameter matrix for an element-wise multiplication layer, which is needed because the range of W_e can go beyond $[-1, 1]$.

TABLE II: MSE and MNIST accuracy for different predictors

	NIA	Mask	RCN	SCN
MSE on random weights	3.21E+0	1.13E+0	2.61E-2	1.51E-2
MSE on trained weights	3.31E+0	1.22E+0	3.75E-2	3.98E-2
Pre-validation accuracy	98.55%	98.39%	98.03%	98.20%
Validation accuracy	28.15%	37.10%	97.80%	97.71%

We train the network to minimize the mean squared error loss, defined as

$$L = \frac{1}{N} \|\hat{W_e} - W_e\|_2^2 \quad (8)$$

where $N = nm$ is the number of elements in W_e, and $\hat{W_e}$ is the estimated effective weight matrix.

B. Scaling Convolutional Network

Scaling convolutional network (SCN) is our convolution approach to regression. Convolution layers are good at capturing spatial patterns in the input with a small number of parameters. On the other hand, SCN is clearly distinguished from convolutional neural networks for image classification; the output is not class labels, but a transformed version of the input, with the same size and data type as the input.

To simplify the process of designing a new network, we leverage the mask idea to capture the spatial correlation, and add convolution layers to compensate for the effect of ReRAM cell values. At the top level, SCN consists of two element-wise scaling layers and a convolutional network in between.

The internal convolutional network is defined to be a stack of n convolutional layers, each with c output channels, except for the last (which has a single channel), where n and c are design parameters. The convolution filter size is fixed to 3×3, and padding and stride are both 1; no pooling layer is used. Activation function, *ReLU*, is used in all layers except the last.

Specifically, the basic block CL_c is a convolutional layer with c output channels. We replicate the basic block n times with the last one having a single channel (i.e., CL_1).

$$\mathbf{CL_c} : X \to Y \text{ where } Y = ReLU(Conv_c(X)) \quad (9)$$

$$\mathbf{SCN_{n,c}} : W_e = CL_1(CL_c(\cdots(CL_c(W \circ U_1)))) \circ U_2 \quad (10)$$

The performance of SCN depends on the value of n and c. Through exploration using a randomly generated dataset for 128×128 array with wire resistance of 1 Ω (more details in Section V-A), we have found that $\text{SCN}_{7,32}$ gives the best balance between performance and model size; further increasing the hyperparameters didn't give much performance improvement. SCN is trained to minimize mean squared error.

Once the regression models are trained, they are integrated into the target DNN's training framework (see Fig. 2c), so that the training of target DNNs can find the best *programmed weight* under *predicted* MVM distortion. During this training, the regression model's parameters are fixed.

V. Experiments

A. Experimental Setup

To evaluate the effectiveness of our proposed technique, we use the BinaryNet framework [12] for MNIST, CIFAR10,

and SVHN datasets.[3] Table I lists the key parameters of the networks and training. Our primary metric is validation accuracy on unseen data (see Section III-A). The baseline accuracy is the test accuracy on GPU, which is the highest we can expect for validation accuracy. For the training under distorted MVM (Step 3 of Fig. 2) we retrain the network from the baseline trained weight, which is commonly called *retraining*. For retraining, we reduce the initial learning rate by 1/8 from that of the baseline training.

We compare the following five cases: NIA, mask, RCN, SCN, and the validation accuracy from Step 2 (referred to as "w/o retraining"). We evaluate 3 different crossbar sizes, 32×32, 64×64, and 128×128. We use the following device parameters, taken from recently fabricated devices [13]: LRS = 1E3 Ω, HRS = 1E6 Ω. The wire resistance per cell (R_w) is varied from 0.1 Ω to 2 Ω. These parameters are very similar to what is used in previous work [10], [11], including the device-to-wire resistance ratio.

To train the IR drop prediction models, we use 50 000 randomly generated crossbar-sized binary data as weight (W). For NIA, we use the trained weights of each target network as described in [10]; thus, the number of data samples differs depending on the network and the crossbar size. For the mask method, we follow the procedure in [11], which uses 100 random data samples. Note that we train the IR drop prediction models again for each device parameter combination (RCA size and wire resistance), but use the same trained model across different networks and between training vs. test. The only exception is the NIA method, which is trained again for each network as well as for each device parameter combination.

B. MVM Computation Accuracy

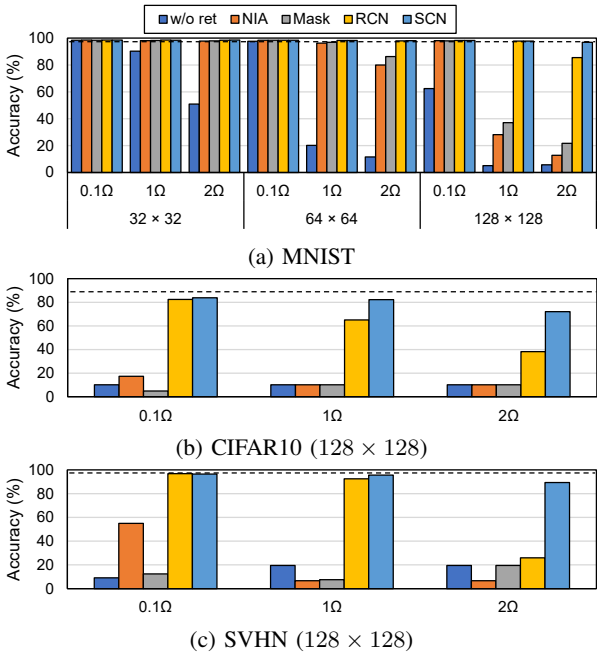
We first evaluate the accuracy of various prediction methods,[4] that is, how close their prediction is to SPICE simulation result. For this experiment we use random weights and inputs (1000 random binary weight matrices and the same number of random binary vectors), as well as trained weights and random inputs. The trained weights are obtained from the MNIST BNN, trained for each prediction method, meaning that the weights are all different depending on the prediction method.

The results summarized in Table II are carried out with $R_w = 1\ \Omega$ and 128×128 RCA. MSE is the mean squared error of RCA output current, averaged over weight matrices. Pre-validation accuracy is the accuracy from Step 3, reported by the MNIST BNN training framework, and thus not representative of the real accuracy under distorted MVM; however, it does indicate how successful the training was.

The table clearly shows that our prediction models are definitely more accurate than the previous methods in terms of MSE (by about 2 orders of magnitude). Consequently, though all methods achieve over 98% pre-validation accuracy, we see stark difference in the validation accuracy they achieve. One interesting thing in the result is that RCN outperforms SCN, very slightly, in this particular case, despite SCN having lower MSE on random weights. This can be explained by the third row, where RCN beats SCN slightly in terms of "MSE on trained weights", suggesting a link between the latter and the accuracy drop.

[3]https://github.com/itayhubara/BinaryNet

[4]Though NIA and mask in the previous work are not intended to be a prediction method, they can be viewed as one.



(a) MNIST

(b) CIFAR10 (128×128)

(c) SVHN (128×128)

Fig. 3: Validation accuracy for different RCA parameters. Dotted lines represent baseline accuracy.

C. Network Training Performance

Fig. 3 shows the validation accuracy for all three networks, where the dotted line represents the baseline accuracy on GPU. Again these results are from Step 4, using unseen test data.

For the MNIST BNN, we explore all combinations of wire resistance values and RCA sizes. From the result we make the following observations. First, the result confirms the severity of the IR drop problem in RCA-based neural networks. In particular, even when wire resistance is as low as 0.1 Ω, 128×128 RCA's MVM computation is so distorted that without retraining it is unusable even for MNIST BNN. Second, all retraining methods including NIA and mask do help train the BNN. For instance, in the case of 64×64 RCA with 1 Ω, the validation accuracy without retraining is only about 20%, but through retraining, all methods recover the baseline-level accuracy. Third, our proposed methods consistently outperform the previous methods. While that is not easy to see for MNIST BNN except for a few difficult cases, the performance difference is very clear in the case of CIFAR10 and SVHN.

The CIFAR10 and SVHN results unequivocally show the superiority of our proposed methods over the previous methods. In particular, the SCN model not only outperforms all the other methods, but it also gives acceptable accuracy where the

TABLE III: CIFAR10 Pre-validation accuracy (%)

R_w	NIA	Mask	RCN	SCN
0.1	86.22	81.77	84.10	84.09
1	84.04	84.29	79.96	82.69
2	85.32	82.20	67.81	78.27

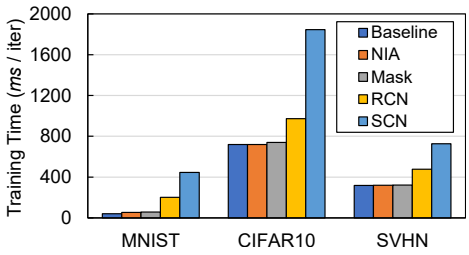


Fig. 4: Training time comparison (per iteration).

previous methods have completely failed. Note that though the accuracy of "w/o retraining" for SVHN is sometimes 20%, it is due to the unbalanced label distribution; for SVHN, 20% accuracy is no better than 10% or less.

To better understand why accuracy drops in CIFAR10, which is the most challenging dataset among the three, we compare the pre-validation accuracy of the prediction methods in Table III. Interestingly, while NIA and mask always achieve over 80% accuracy *before validation*, RCN's accuracy varies with the RCA parameter, reaching below 70% in one case. SCN also shows under 80% accuracy with $R_w = 2\ \Omega$. This result suggests that in the case of NIA and mask, the prediction model is clearly the culprit. In the case of RLN and SCN, we argue that it is the combination of CIFAR10 training and model imperfection. Model imperfection, since the gap between pre-validation accuracy and validation accuracy can be attributed to the inaccuracy of the IR drop prediction model. CIFAR10 training, since even the pre-validation accuracy didn't reach the baseline level in some cases.

D. Time Overhead Comparison

Fig. 4 compares the training time of BNNs using various IR drop mitigation methods. The BNN framework is implemented in Torch7, and training is done using a GPU on a system with Intel Xeon CPU E5-2630 v4 and Nvidia GPU GeForce GTX 1080Ti. The total training time depends on the number of iterations and the number of epochs as well. Although RCN and SCN take more time to evaluate additional neural network within BNN training, it is justifiable by the definite performance boost as shown in Fig. 3. Also the overhead is negligible compared to running IR drop simulation during training.

E. Noise Sensitivity

Table IV shows the test validation accuracy for MNIST with $R_w = 1\ \Omega$ and 64×64 RCAs, when 10% Gaussian noise is added to weight parameters during both retraining and validation. Our result shows that our proposed methods are robust to noise, and the accuracy degradation is small, and definitely no worse than the previous methods.

TABLE IV: Accuracy with 10% Gaussian noise in weights

	NIA	Mask	RCN	SCN
Validation accuracy (%)	94.57	95.47	97.85	97.55
Degradation (%p)	-3.84	-2.97	-0.6	-0.87

VI. Conclusion

We presented a novel method to incorporate the IR drop problem during BNN training with a negligible overhead. Compared to hardware methods (e.g., new device/selector material, error compensating circuitry), our training method is essentially free, and applicable on top of any hardware methods. Our experimental results demonstrate that while the IR drop problem renders many passive ReRAM crossbar configurations unsuitable for DNN inference, our proposed method can extend the range of usable configurations significantly, achieving near-baseline level test validation accuracy with MNIST and SVHN BNNs, and a significant boost with CIFAR10 BNN.

We see many paths for future work. Our technique can be applied to multi-bit networks. ReRAMs have many nonidealities including variability, stochastic noise, and permanent faults, with some of them very damaging. Training in the presence of unpredictable nonidealities is left for future work.

References

[1] Q. Xia and J. J. Yang, "Memristive crossbar arrays for brain-inspired computing," *Nature materials*, vol. 18, no. 4, pp. 309–323, 2019.

[2] 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.

[3] K. Chellapilla *et al.*, "High Performance Convolutional Neural Networks for Document Processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, Oct. 2006.

[4] 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, pp. 448–456.

[5] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an fpga," in *IEEE IPDPS Workshops*, May 2017, pp. 98–105.

[6] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," ser. ISCA '16, Piscataway, NJ, USA, 2016, pp. 27–39.

[7] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 541–552.

[8] Y. Cai *et al.*, "Long live time: Improving lifetime for training-in-memory engines by structured gradient sparsification," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 107:1–107:6.

[9] X. Qiao *et al.*, "Atomlayer: A universal reram-based cnn accelerator with atomic layer computation," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18, 2018, pp. 103:1–103:6.

[10] Z. He *et al.*, "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, 2019, p. 57.

[11] M. E. Fouda *et al.*, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 704–716, 2019.

[12] I. Hubara *et al.*, "Binarized neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Curran Associates Inc., 2016, pp. 4114–4122.

[13] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.