Digital Offset for RRAM-based Neuromorphic Computing: A Novel Solution to Conquer Cycle-to-cycle Variation

Ziqi Meng[1], Weikang Qian[1,3,4], Yilong Zhao[2], Yanan Sun[2], Rui Yang[1], Li Jiang[2,3]
[1]University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, Shanghai, China
[2]School of Electronic, Information, and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
[3]MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai, China
[4]State Key Laboratory of ASIC & System, Fudan University, Shanghai, China

Abstract*—Resistance variation in memristor device hinders the practical use of resistive random access memory (RRAM) crossbars as neural network (NN) accelerators. Previous fault-tolerant methods cannot effectively handle cycle-to-cycle variation (CCV). Many of them also use a pair of positive-weight and negative-weight crossbars to store a weight matrix, which implicitly enhances the fault tolerance but doubles the hardware cost. This paper proposes a novel solution that dramatically reduces the NN accuracy loss under CCV, while still using a single crossbar to store a weight matrix. The key idea is to introduce digital offsets into the crossbar, which further enables two techniques to conquer CCV. The first is a variation-aware weight optimization method that determines the optimal target weights to be written into the crossbar; the second is a post-writing tuning method that optimally sets the digital offsets to recover the accuracy loss due to variation. Simulation results show that the accuracy maintains the ideal value for LeNet with MNIST and only drops by 2.77$\%$ over the ideal value for ResNet-18 with CIFAR-10 under a large resistance variation. Moreover, compared to state-of-the-art fault-tolerant methods, our method achieves a better NN accuracy with at least 50% fewer crossbars.

***Index Terms*—RRAM, unreliable device, cycle-to-cycle variation, neural network, digital offset**

I. Introduction

Resistive random access memory (RRAM) crossbar, composed of memristors [1], is a promising choice for realizing neural network (NN) accelerators, as it enables efficient *vector-matrix multiplication (VMM)*, a dominant operation of NNs. Some practical accelerator architectures based on RRAM crossbar have been proposed, such as PRIME [2] and ISAAC [3]. A big challenge, however, is that the device non-idealities, particularly the resistance variation, degrade their accuracy.

A memristor mainly suffers from two types of variations, device-to-device variation (DDV) and cycle-to-cycle variation (CCV). The former, caused by fluctuations during fabrication, makes each device perform differently, while the latter leads to different final resistances for the same device and the same target state due to fluctuations in the driving circuits [4].

Many methods have been proposed to tolerate variations and other faults in crossbar. Programming-based methods that iteratively apply pulses until the convergence to a target resistive range are proposed in [5], [6]. New training methods are designed to enhance the robustness of accelerators, including re-designing the objective function in training [7], [8] and training the NNs with variations [9]. Some other methods overcome hardware defects by properly mapping the weights to the devices [7], [10]–[12].

However, these existing works still have the following issues.

- The CCV issue has not been effectively solved yet. Although the repeated programming method is a potential solution [5], [6], it needs to program the device many times, which shortens the device lifetime. Some training-based methods can only handle small CCV for simple NNs [7]–[9]. Many mapping-based methods assume that the resistance deviation of a device measured after different programming cycles does not change [7], [10]–[12]. Thus, they inherently do not take CCV into consideration.
- It is hard to maintain accuracy for complex NNs under large device variation with acceptable training and testing overhead. On the one hand, some training-based methods may not converge under large device variation [7], [9]. On the other hand, some mapping-based method requires testing all the devices *for each resistance state*, causing a considerable testing overhead [7], [10]–[12].
- Many fault-tolerant methods are based on two-crossbar architecture, where one crossbar stores the positive weights, while the other stores the negative ones [10]–[12]. A less costly architecture, the one-crossbar architecture, which uses a single crossbar to store a weight matrix, has weaker fault tolerance, preventing its widespread applications. How to improve the fault tolerance of the one-crossbar architecture remains to be an open problem.

In this work, we propose a novel method to address all the above challenges. The key idea is to introduce tunable digital offsets, each shared with multiple weights in a crossbar. Even if the actual weights in the crossbar are different from the target values, we can recover the NN accuracy loss by properly setting the digital offsets. We call this technique *post-writing tuning (PWT)*. It turns out that by PWT, we can still use the one-crossbar architecture to achieve a high enough NN accuracy. Furthermore, our method only requires writing the crossbar once and then testing each actual conductance value once. This reduces the testing overhead compared to some previous methods that need to test each device for each target state. Besides, since the digital offset is determined after measuring the actual conductance value, the total conductance deviation, which includes both the DDV and the CCV, is naturally taken into account. Note that a recent work also introduces a post-processing-based compensation method for RRAM crossbars [13]. However, it targets at stuck-at-faults, different from our target, resistance variation. Furthermore, it compensates each faulty device individually, causing a large hardware overhead. In contrast, our method compensates a group of devices with a single digital offset, thus, only introducing a small overhead. Moreover, we propose sophisticated

This work is supported in part by the National Key R&D Program of China under Grants 2020YFB2205501 and 2018YFB1403400, the State Key Laboratory of ASIC & System Open Research Grant 2019KF004, National Natural Science Foundation of China Grant 61834006, Shanghai Science and Technology Committee Grant 18ZR1421400, Shanghai Sailing Program Award 19YF1424900, SJTU Major Frontier Program Grant 2019QYA01, and a grant from Huawei Technologies Co., Ltd. Corresponding authors: Weikang Qian, Yanan Sun, and Li Jiang. Emails: {qianwk, sunyanan, ljiang_cs}@sjtu.edu.cn.

algorithms to fully utilize the low-overhead compensation architecture.

The main contributions of this work are as follows.

- We propose to introduce digital offsets into the RRAM crossbar-based NN accelerators. It helps conquer both the DDV and the CCV and allows the adoption of the less costly one-crossbar architecture.
- With the digital offsets, we propose an associated PWT technique to improve the NN accuracy under device variations. A simple training approach is proposed to derive the optimal digital offsets for the PWT.
- The digital offsets also give us freedom in determining the target weights to be written into the crossbar. We propose a related technique called *variation-aware weight optimization (VAWO)* to decide a set of target weights that maximizes the NN accuracy.
- An enhancement to VAWO that decides whether complemented weights should be set as the targets is proposed to further improve the NN accuracy.
- The architecture support for the digital offset is proposed.

The experimental results show that by applying the proposed method, the accuracy of LeNet improves to the ideal value of 99.17% using single-level cell (SLC)-based RRAM crossbar architecture. The accuracy of ResNet-18 is still above 90% using 2-bit multi-level cell (MLC)-based RRAM architecture even under a large variation. Compared to the basic one-crossbar architecture, the power overhead of our proposed design is only 2.4%-6.9%. Compared to the state-of-the-art fault-tolerant methods [9], [12], our method achieves a better NN accuracy with at least 50% fewer crossbars.

II. Background: RRAM-based NN Accelerator

The basic operation in NN is VMM of the form $\vec{y} = \vec{x}A$, where $\vec{y}$ and $\vec{x}$ are the output and input vectors, respectively, and A is the weight matrix. As shown in Fig. 1(a), such a VMM operation can be efficiently implemented by an RRAM crossbar, where the conductance of the device at row i and column j is set as $A_{i,j}$, the entry at row i and column j of matrix A, and the voltage on wordline i is set as x_i, the i-th entry in vector $\vec{x}$. By the Kirchhoff's Law, the output current of bitline j in the crossbar is $\sum_i x_i \cdot A_{i,j}$, which equals y_j, the j-th entry in vector $\vec{y}$.

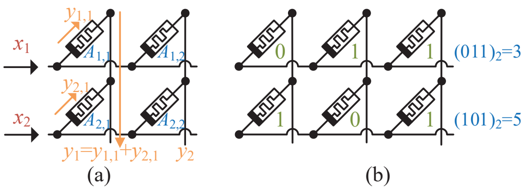

Fig. 1. An illustration on crossbar. (a) A crossbar with analog conductance values. (b) A crossbar using 3 SLCs to represent a weight.

However, due to current technology limitation, we cannot precisely set the conductance of a memristor device to an arbitrary value. In practice, a device is programmed to one of the multiple target resistance states. The basic version, *SLC*, has just two states, a *high resistance state (HRS)* and a *low resistance state (LRS)*. Advanced technology can fabricate *MLC*, which has a few more *middle resistance states (MRSs)* [14]. A typical MLC is the 2-bit MLC, which has 4 resistance states. In the following, we describe our method assuming that SLCs are used. However, it can be easily extended for MLCs.

For an SLC, it encodes 0 (1) if it is in HRS (LRS), as a low (high) current is generated. Thus, an SLC can represent 1 bit. A practical RRAM-based accelerator has weights encoded as binary numbers and further represents the binary numbers by SLCs or MLCs. Fig. 1(b) shows an example of representing each weight by 3 SLCs. The currents from the 3 columns will be fed to sample-and-hold (S&H) circuits and then converted to 3 digital signals by an analog-to-digital converter (ADC) column by column. Finally, the 3 signals will be shifted and added to get the final value by a shift-and-add (S+A) unit [3].

There exist two typical RRAM-based architectures, the two-crossbar architecture and the one-crossbar architecture. The former uses a crossbar to store the positive weights and another to store the negative weights [2]. The latter stores all the weights in just one crossbar through a weight shift, leading to significant hardware cost reduction [3]. Due to this benefit, we focus on the one-crossbar architecture in this paper. The underlying architecture we use is ISAAC [3].

In order to handle negative weights, the one-crossbar architecture applies a shift to make all weights non-negative [3]. For example, weights initially in the range $[-120, 135]$ are shifted to the range $[0, 255]$ by adding each with 120. After the calculation by the crossbar, we should subtract $120 \cdot \sum x_i$ from the result to get the final output. Due to this technique, in the following, we assume that the weights in the crossbar are all non-negative.

III. Proposed Techniques

In this section, we describe the proposed method. We begin with the motivation and an overview of the proposed method, followed by the detailed discussion of each component.

A. Motivation and Overview

Resistance variation makes the actual weight written in the crossbar differ from the target weight. A simple solution is to compensate the weight deviation by an offset. For example, if the target weight is 10, but the actual weight in the crossbar is 8.5, an offset of 1.5 should be applied for compensation. The basic requirement for the offset is that *it can be configured to a specified value accurately*. Thus, we resort to digital register for storing the offset and hence, call the offset *digital offset*. However, if each weight is accompanied by a digital offset, we need many registers, causing a large overhead. To reduce the overhead, we share an offset with multiple weights.

As a basic version, a digital offset is shared with all the weights in a column in the matrix. If the actual values of these weights in the crossbar are $v_1, \dots, v_n$, then with a common offset b, they become $v_1 + b, \dots, v_n + b$. Then, the dot product over the inputs x_i's and the corrected weights $(v_i + b)$'s is

$$z = \sum_i x_i (v_i + b) = \sum_i x_i v_i + b \sum_i x_i. \qquad (1)$$

Since all the weights share the same offset, but their deviations can be different, it is impossible to find an offset that can *exactly compensate the deviations for all the weights*. Fortunately, since NN is inherently error tolerant, we only need to choose a proper offset b so that the NN output is close to the expected one.

Eq. (1) also gives an implementation of the compensation by the digital offset. Its first term is the output from the crossbar, which involves the actual conductance deviations. The second term can be realized by first summing over all the x_i's and then multiplying the sum with the offset b stored in the register.

Fig. 2 illustrates the digital offset technique. Fig. 2(a) shows the original weight matrix and the expected outputs. Due to resistance variation, some deviations are injected into the weight matrix, as shown in Fig. 2(b). This causes errors in the final outputs. Fig. 2(c) shows the implementation of the digital offset. Each column has a digital offset shared with all the weights in the column. To compensate for the final output, the inputs in the column are first summed and then the sum is multiplied with the digital offset. This leads to the compensation of $(3{+}0{+}1)\cdot(-0.3){=}-1.2$ for the 1st column and -1.6 for the 2nd column. With the compensation, the output errors are significantly reduced compared to those in Fig. 2(b).

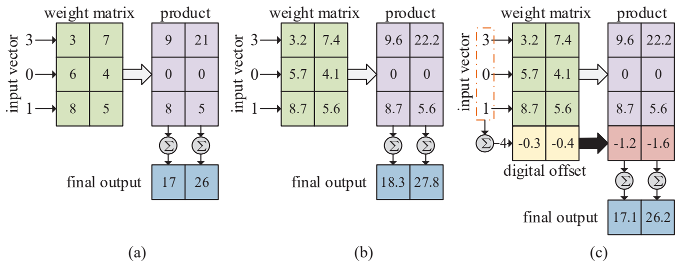

Fig. 2. An illustration of the proposed digital offset technique. (a) A VMM operation in the ideal case. (b) A VMM operation under resistance variation. (c) A VMM operation with the compensation by the digital offset technique.

In practice, to save power and improve the readout accuracy, only a limited number of wordlines in a crossbar should be activated in each cycle [15], [16]. In a 128 $\times$ 128 RRAM crossbar, for example, only 16 wordlines are activated in each cycle. This gives us an opportunity to share an offset with fewer devices, i.e., only those devices on the activated wordlines in a cycle. At a cost of more cycles to complete a VMM operation, it enables a finer-grained sharing, beneficial to further NN accuracy improvement. In the following, we assume that an offset is shared with m weights, where m can be any multiple of the number of wordlines activated in each cycle. We refer to m as the *sharing granularity*.

Since the above technique determines the digital offsets after all the devices are programmed, we call it *post-writing tuning (PWT)*. A related important problem is *how to choose the optimal digital offsets*. We propose a network training procedure to solve this problem.

With the aid of digital offsets, a weight to be written into the crossbar need not equal the original one in the NN. For example, instead of writing $3, 6, 8$ into column 1 of the crossbar shown in Fig. 2(a), we can write $1, 4, 6$. The weight difference can be compensated by properly setting the digital offset later. Thus, the digital offset technique offers flexibility to choose the target value to be written in the crossbar. Note that for memristor devices, different target values lead to different amounts of deviation after writing. To fully exploit the flexibility, another critical problem is *how to decide the optimal weights to be written in the crossbar to maximize the expected NN accuracy*. We propose a *variation-aware weight optimization (VAWO)* technique to solve this problem. Moreover, we introduce a related weight complement technique that also considers the complemented weight. It provides extra optimization space to further improve the NN accuracy.

In summary, the introduction of digital offset enables two important techniques to conquer variations, PWT and VAWO. They can either work alone or be combined to complement each other. VAWO determines the weights to be written in the crossbar. Thus, it is applied before weight writing, using *priori knowledge* on the distribution of the conductance deviation. In contrast, PWT is used after weight writing, exploiting the *posteriori knowledge* on the actual conductance deviation.

Next, we will elaborate VAWO and the related weight complement technique in Sections III-B and III-C, respectively. Then, we describe PWT in Section III-D. Finally, we show the architecture support for the digital offsets in Section III-E.

B. Variation-Aware Weight Optimization

In this section, we elaborate VAWO. A key observation enabling VAWO is that with digital offsets, a weight to be written into the crossbar need not equal the one in the NN. Furthermore, due to resistance variation, a target weight may not equal the actual weight in the crossbar. Considering the above situations and to facilitate our discussion, we first define the following four types of weights.

- **Network target weight (NTW)**: An NTW w is the weight of the NN obtained after training. It is the target weight we try to achieve.
- **Crossbar target weight (CTW)**: A CTW v is the target weight to be written into the crossbar. With the digital offset, it need not equal NTW.
- **Crossbar real weight (CRW)**: A CRW V is the actual weight stored in the crossbar when we try to write the CTW v. The difference $V - v$ is caused by the non-idealities including the resistance variation.
- **Network real weight (NRW)**: An NRW is the finally achieved weight for the NN. It equals the sum of the CRW and the digital offset.

Fig. 3 shows an example of the four types of weights. Each value in a green box is a bit and that in a purple or a pink box is the corresponding decimal value. The value in the purple box in Fig. 3(a) is an NTW, while the one in Fig. 3(b) is a CTW obtained by VAWO for that NTW. Note that the CTW differs from the NTW. In Fig. 3(c), some random variations are injected into the bits of the CTW, which leads to the CRW of 2.1. The value in the pink box is the NRW for the CRW, assuming an offset of 1.

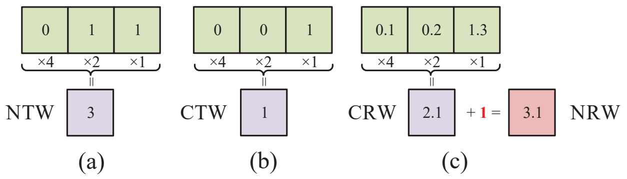

Fig. 3. An illustration on the four types of weights. (a) An NTW. (b) A CTW obtained by VAWO. (c) A CRW and an NRW with $b = 1$.

Given the flexibility provided by a digital offest, the CTWs sharing that offset have many choices. Different choices of the CTWs cause different deviations for the CRWs and hence, the final NRWs sharing that offset. This eventually leads to different NN accuracies. For each set of weights sharing an offset, VAWO tries to determine their optimal CTWs that maximize the accuracy. To simplify the illustration, we assume that the sharing granularity is $m = 2$. We consider a particular pair of weights w_1 and w_2 sharing an offset. In order to find the optimal CTWs for w_1 and w_2 that maximize the NN accuracy, we focus on minimizing the loss function L on w_1 and w_2.

Assume that after training, the NTWs for w_1 and w_2 are w_1^* and w_2^*, respectively. Assume that the CTWs are v_1 and v_2, respectively. Our task is to determine a good choice for v_1 and v_2. Due to the random device variation, the CRWs do not equal CTWs. Denote the CRWs as V_1 and V_2, which are mutually independent random variables. We assume that

$$V_1 = R(v_1), \quad V_2 = R(v_2),$$

where R is a function mapping the CTW into the CRW. Suppose the digital offset is set to b. Then, by definition, the NRWs, denoted as W_1 and W_2, are

$$W_1 = V_1 + b = R(v_1) + b, \quad W_2 = V_2 + b = R(v_2) + b. \quad (2)$$

They are the actual weights of the NN. In summary, considering the digital offset and the device variation, the loss function becomes $L(W_1, W_2)$, which is also a random value. Our target is to minimize the loss $L(W_1, W_2)$ in a statistical sense. A mathematically friendly choice is to minimize $E[L^2(W_1, W_2)]$, where $E[X]$ is the expectation of the random variable X.

Define

$$\Delta W_1 = W_1 - w_1^*, \quad \Delta W_2 = W_2 - w_2^*. \tag{3}$$

ΔW_i ($i = 1, 2$) gives the difference between the NRW and the NTW for w_i. Clearly, ΔW_1 and ΔW_2 are mutually independent random variables. By the first-order derivative, we have

$$L(W_1, W_2) \approx L(w_1^*, w_2^*) + \sum_{i=1}^{2} \frac{\partial L(w_1^*, w_2^*)}{\partial w_i} \Delta W_i.$$

For simplicity, we choose the CTWs v_1 and v_2 such that

$$E[W_1] = w_1^*, \quad E[W_2] = w_2^*. \tag{4}$$

By Eqs. (3) and (4), we have $E[\Delta W_1] = E[\Delta W_2] = 0$. Given that ΔW_1 and ΔW_2 are independent, we further have

$$E[L^2(W_1,W_2)] \approx L^2(w_1^*,w_2^*) + \sum_{i=1}^{2} \left(\frac{\partial L(w_1^*,w_2^*)}{\partial w_i} \right)^2 Var[\Delta W_i].$$

Since $L(w_1^*, w_2^*)$ is a constant, to minimize $E[L^2(W_1, W_2)]$, we only need to focus on the 2nd term in the above equation. By Eqs. (2) and (3) and the fact that w_1^*, w_2^*, and b are deterministic, we have $Var[\Delta W_1] = Var[R(v_1)]$ and $Var[\Delta W_2] = Var[R(v_2)]$. Then, the optimization target becomes

$$\sum_{i=1}^{2} \left(\frac{\partial L(w_1^*, w_2^*)}{\partial w_i} \right)^2 Var[R(v_i)]. \tag{5}$$

Note that the gradient $\partial L / \partial w_i$ in Eq. (5) can be obtained by running inference on the training dataset. It equals the mean of the gradients of all the training samples.

By Eqs. (2) and (4), we have

$$E[R(v_i)] + b = w_i^*, \text{ for } i = 1, 2, \tag{6}$$

which serves as the constraints. In summary, VAWO can be formulated as an optimization problem with the objective function shown in Eq. (5) and the constraints shown in Eq. (6). The variables are v_1, v_2, and b. The optimal solution for v_i is the CTW for weight w_i.

To set up the optimization problem, we also need to know $E[R(v)]$ and $Var[R(v)]$ for each CTW v. This can be obtained from a statistical testing. Assume that a CTW v has n bits. For each CTW v, K random sets of n memristors are selected. For each set, it is programmed with the CTW v for J times and the final CRWs are measured. After collecting KJ CRWs for the CTW v, we can calculate $E[R(v)]$ and $Var[R(v)]$. By iterating over all CTWs v's, we can finally build a look-up table (LUT) on $E[R(v)]$ and $Var[R(v)]$ for each possible v.

The above optimization problem is formulated for each set of weights sharing an offset. For each set, the problem can be solved by iterating over all possible choices for the offset b. With b as an n-bit binary number, the total number of choices is 2^n. For each choice of b, by Eq. (6) and the LUT for $E[R(v)]$, we can reversely solve for all the v_i's. Then, by the LUT for $Var[R(v)]$, we can get the objective value for this choice. Finally, we obtain the minimum objective value over all the offset choices. The corresponding v_i's are the optimal CTWs.

Like NN training, VAWO is a one-time process, but its runtime is much less than the NN training time. For example, for LeNet, when the offsets are 8-bit, the runtime for VAWO is 19.7s, which is only 4.3% of the training time.

C. Weight Complement

To further optimize the objective function of VAWO, we introduce a *weight complement* technique. It provides another choice, i.e., the complemented value, for each weight and hence, expands the solution space. For an original n-bit weight w, its complement is $\overline{w} = (2^n - 1) - w$. Given NTWs w_i^*'s sharing the same offset and inputs x_i's, suppose that we want to compute their dot product $z = \sum_i w_i^* x_i$. Since by definition, $w_i^* = 2^n - 1 - \overline{w_i^*}$, we have

$$z = \sum_i w_i^* x_i = \sum_i (2^n - 1 - \overline{w_i^*}) x_i = (2^n - 1) \sum_i x_i - \sum_i \overline{w_i^*} x_i.$$

The above equation means that to calculate z, we can instead first calculate $z' = \sum_i \overline{w_i^*} x_i$ using the complemented weights $\overline{w_i^*}$'s as the NTWs. Then, we calculate $(2^n - 1) \sum_i x_i - z'$, which is supported by ISAAC [3].

In summary, to perform a dot product, the NTWs sharing the same offset have two choices: their original form and the complemented form. Correspondingly, we can set up two optimization problems. The first is the original VAWO formulation, while the second is a modified version by replacing each w_i^* in Eq. (6) with $\overline{w_i^*}$. The NTWs will be complemented if the second problem has a better optimal solution than the first, and the module calculating $(2^n - 1) \sum_i x_i - z'$ from ISAAC should be enabled to post-process the dot product.

D. Post-Writing Tuning on Digital Offset

In PWT, each device is first tested to obtain its actual conductance. With such information known, we can further optimize the digital offset values to improve the NN accuracy.

Assume the matrix has N rows and the input vector is $(x_1, \ldots, x_N)$. Assume the CRWs of a particular column is $V_i, \ldots, V_N$. Given sharing granularity m, let $k = N/m$. Then, k represents the number of the sets of weights sharing the same offset in the column. We denote the offset shared by the weights $V_{im+1}, \ldots, V_{(i+1)m}$ ($0 \le i \le k-1$) as b_i. Then, the output of that column is

$$\begin{aligned} z &= \sum_{i=0}^{k-1} \sum_{j=1}^{m} (V_{im+j} + b_i) \cdot x_{im+j} \\ &= \sum_{i=0}^{k-1} \sum_{j=1}^{m} V_{im+j} x_{im+j} + \sum_{i=0}^{k-1} b_i \sum_{j=1}^{m} x_{im+j}. \end{aligned} \tag{7}$$

In PWT, V_i's are known and the unknowns to be optimized are b_i's. We can now treat the original NN as a new one with trainable parameters as b_i's. Thus, we can apply the back propagation algorithm [17] to obtain optimal b_i's. By Eq. (7), it is easy to derive the correction Δb_i for b_i as

$$\Delta b_i = -\eta \frac{\partial L}{\partial b_i} = -\eta \frac{\partial L}{\partial z} \frac{\partial z}{\partial b_i} = -\eta \frac{\partial L}{\partial z} \sum_{j=1}^{m} x_{im+j}, \tag{8}$$

where η is the learning rate. When the training process guided by Eq. (8) finishes, we obtain the optimal digital offsets.

E. Architecture and Pipeline

We can simply modify the ISAAC [3] architecture to support the digital offset, which is shown in Fig. 4. The modification is shown in the shaded box, which contains extra registers for the offset storage and extra adders and multipliers to implement the term $b \sum_i x_i$ in Eq. (1). The final product produced by the multiplier is added into the output from the crossbar through the adder in the S+A unit. If a crossbar has S rows and stores l

columns of weights in the matrix, then for a sharing granularity of m, the number of extra registers for each crossbar is

$$H = Sl/m. \tag{9}$$

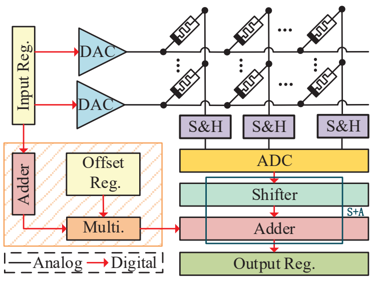

Fig. 4. The architecture with the additional support for the digital offsets.

To support the digital offset, we also slightly modify the pipeline of ISAAC by adding a sum-and-multiply (Sum+Multi) operation. As we will show in Section IV-B2, it can be integrated into the ISAAC pipeline without increasing the latency.

IV. Experimental Results

In this section, we evaluate our proposed method by simulation. We use two test cases: LeNet with MNIST dataset and ResNet-18 with CIFAR-10 dataset. The two NNs are trained with Pytorch. Their ideal accuracies are 99.17% and 94.14%, respectively. We use the cross-entropy loss function.

The weights and the inputs are quantized to 8 bits. The digital offsets are also 8-bit. We consider both SLC and 2-bit MLC to build the RRAM crossbar. Considering the DDV and the CCV, we model the actual conductance as a log-normal random variable with respect to the nominal value [4]. Specifically, the mapping function from CTW to CRW is $V = R(v) = ve^{\theta}$, where θ is a normal random variable with zero mean and the standard deviation $\sigma \in [0.2, 1.0]$. Due to the CCV, the CRWs take different values in different programming cycles even for the same CTWs. Thus, each experiment is repeated 5 times with different CRWs each time and the average result is reported. We also consider the finite ON/OFF ratio of memristors, which is set to 200. The crossbar size is 128×128. We consider 3 sharing granularities m: 16, 64, and 128. For each m, we assume only m wordlines are activated in each cycle. For simplicity, we denote VAWO with the weight complement technique as *VAWO**. For comparison, we also include a *plain* scheme, which sets the CTWs as the NTWs without any offsets introduced.

A. Accuracy

In this section, we compare the accuracies of the proposed methods. Figs. 5(a) and 5(b) show the accuracies of different methods and sharing granularities for LeNet and ResNet, respectively, with SLCs and $\sigma = 0.5$. Fig. 5(c) shows the results of ResNet using 2-bit MLCs with varying σ. The red line at the top of each figure indicates the ideal accuracy and the blue one near the bottom indicates the accuracy of the plain scheme.

1) Performance of VAWO and the Weight Complement Technique: As shown in Fig. 5(a), the accuracy of the plain scheme for LeNet is only 12.05%. With VAWO, the accuracy rises to 88.48% for sharing granularity $m = 16$. With VAWO*, it further rises to 95.84%. When m increases to 128, the accuracy of VAWO drops, due to the less flexibility of the coarser sharing granularity. For VAWO*, however, the accuracy for $m = 128$ is almost the same as that for $m = 16$, showing the benefit of the weight complement technique. However, as Fig. 5(b) shows, for more complex NN like ResNet, using VAWO* alone is still not enough in recovering the accuracy loss.

2) Performance of PWT: For LeNet, the accuracies by PWT alone for both $m = 16$ and 128 almost match the ideal value, as shown in Fig. 5(a). For ResNet, PWT is ineffective, as shown in Fig. 5(b). The reason is that without properly setting the CTWs, the NRWs are far away from the NTWs. Thus, it is hard to find an offset to well compensate the differences between NRWs and NTWs for all the weights sharing that offset.

3) Performance of Overall Optimization: To overcome the above limits, we combine VAWO* and PWT together. For LeNet, the accuracy reaches the ideal value as shown in Fig. 5(a). For ResNet using SLCs, as shown in Fig. 5(b), the accuracy improves to 91.37% with $m = 16$, which implies a drop of only 2.77%. To save the hardware cost, 2-bit MLCs can be used in crossbar to represent weights. In Fig. 5(c), we show the accuracies of combining VAWO* and PWT together on ResNet using 2-bit MLCs. Despite the higher sensitivity to variations of MLCs [12], the accuracy for $m = 16$ is still over 90% even with $\sigma = 0.7$. For a coarser sharing granularity $m = 128$, the accuracy is still close to 80% even with $\sigma = 1.0$.

B. Area and Power Consumption

In this section, we study the area and power of the proposed architecture. We compare it with the one-crossbar architecture ISAAC. Same as ISAAC, it uses 2-bit MLCs.

1) Saving on Total Device Reading Power: Device reading power is an important part of the total power consumption since reading is repeated in the inference process. The power for reading different resistance states is different. Since VAWO* changes the CTWs, it modifies the resistance states in the crossbar and hence, may change the total device reading power. Table I lists the relative total device reading power by VAWO* to that by the plain scheme for the two test cases. We can see that VAWO* reduces the total device reading power. This is because VAWO* usually reduces the weight values in the crossbar and hence, sets more devices to the higher resistance states with less reading power consumption.

TABLE I
Relative reading power by VAWO* to that by the plain scheme.

	$m = 16$	$m = 128$
LeNet + MNIST	68.87%	79.95%
ResNet + CIFAR-10	57.61%	72.24%

2) Total Overhead: Although the total device reading power reduces, the proposed architecture has some overhead due to the extra adders, multipliers, and registers to support digital offset. In this section, we estimate the total hardware overhead.

In ISAAC, 1 bit of the input is fed into the crossbar in each cycle. Thus, each adder adds m 1-bit numbers. Since the sum of m 1-bit number is no more than 128 and the digital offset has 8 bits, we only need an 8-by-8 multiplier. We choose a Wallace tree multiplier [18]. To reduce the overhead, the multiplier is shared among all columns in a crossbar in a time-multiplexed fashion, while the adder is not. We synthesize the adder and the multiplier by Synopsys Design Compiler with Nangate 45nm open cell library [19]. Then, the results are scaled to 32nm for a fair comparison to ISAAC. By the simulation data, the delay of the Sum+Multi operation does not exceed the clock period of ISAAC, 100ns. Thus, this operation can be integrated into the ISAAC pipeline without increasing the latency.

By Eq. (9), each crossbar needs 256 and 32 offset registers for $m = 16$ and 128, respectively. We assume that the offset

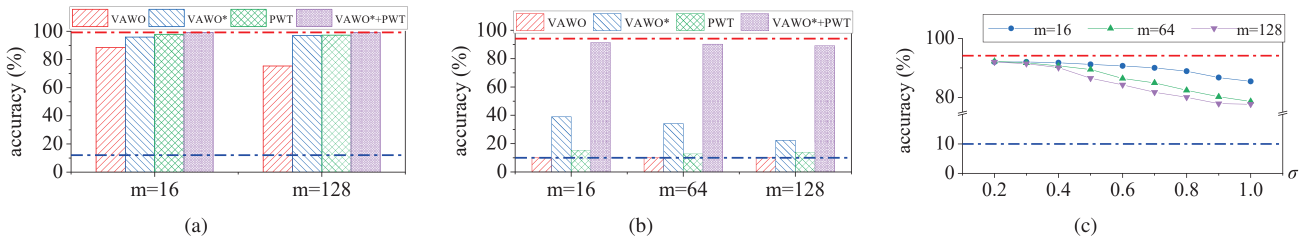

Fig. 5. NN accuracies of different methods and sharing granularities on (a) LeNet and (b) ResNet using SLCs with $\sigma = 0.5$. (c) NN accuracies of using VAWO* and PWT together on ResNet for different σ's using 2-bit MLCs.

registers are built by SRAMs and we use the data from [3] to estimate their overhead in area and power. Combining the overhead of the extra adders, multipliers, and registers with the reading power reduction for ResNet, we obtain the total area and power overheads of a tile [3], which are listed in Table II. Compared to a baseline ISAAC tile, the total overheads in area and power are 13.3% and 2.4%, respectively, for $m = 16$. For $m = 128$, the overheads rise to 17.2% and 6.9%, respectively. This is because as m increases, the cost of the adders increases, which outpaces the reduction of the offset registers.

TABLE II
TOTAL OVERHEAD IN AN ISAAC TILE.

	$m = 16$		$m = 128$	
	Area/mm^2	Power/mW	Area/mm^2	Power/mW
ISAAC Tile	0.372	330	0.372	330
Total Overhead	0.049 (13.3%)	8.05 (2.4%)	0.064 (17.2%)	22.77 (6.9%)

C. Comparison with Other Methods

In this section, we compare our work with three state-of-the-art methods listed in Table III. DVA trains NNs with variations [9]. Priority mapping (PM) optimizes mapping of each weight with unary coding [12]. The final method, DVA+PM, applies DVA first, followed by PM. All the methods are applied to CIFAR-10 dataset. In this experiment, for a fair comparison to [12], our method is applied to VGG-16. The comparison results are listed in Table III. Note that the accuracy losses of the three previous methods are taken from [9] and [12].

TABLE III
COMPARISON AMONG DIFFERENT METHODS.

	DVA [9]	PM [12]	DVA+PM [12]	This work
Network	AlexNet	VGG-16	VGG-16	VGG-16
Variation σ	0.5	0.8	0.8	0.8
Accuracy loss	13%	12.02%	5.48%	4.94%
Crossbar number	2	2.5	2.5	1

1) Accuracy: As shown in Table III, our method has a smaller accuracy loss than DVA even for a more complex NN and a larger variation. Compared to PM and DVA+PM, our method is also better. It should be noted that PM and DVA+PM cannot well handle CCV. Thus, when CCV is considered, we expect that their accuracy losses will further increase.

2) Crossbar Number: DVA is based on one-crossbar architecture, and it uses 8 SLCs to represent a weight [9]. PM and DVA+PM are based on two-crossbar architecture, and they use 10 2-bit MLCs to represent a weight [12]. Our method uses 4 2-bit MLCs to represent a weight. As the number of crossbars needed is roughly proportional to the number of devices used to represent a weight, we derive the normalized crossbar number for these methods in Table III. The crossbar number of our method is the normalization baseline. It shows that our method achieves a better accuracy with at least 50% fewer crossbars.

V. Conclusions

In this work, we propose a novel solution to conquer resistance variation of RRAM crossbar-based NN accelerators. The key idea is to introduce a tunable digital offset shared with multiple weights. It further enables two important techniques, VAWO and PWT. VAWO sets the optimal target weights to be written into the crossbar, while PWT further fine-tunes the digital offsets based on the actual conductances in the crossbar. The proposed method achieves excellent fault tolerance, while still using a less costly one-crossbar architecture. Thus, introducing digital offset is a promising low-cost solution for addressing the variation problem of RRAM crossbar-based NN accelerators. The proposed method is orthogonal to many existing training-based methods such as DVA. Our future work will explore how to combine them together to further reduce the NN accuracy loss due to device resistance variation.

References

[1] D. B. Strukov *et al.*, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.
[2] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA*, 2016, pp. 27–39.
[3] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016, pp. 14–26.
[4] A. Grossi *et al.*, "Fundamental variability limits of filament-based RRAM," in *IEDM*, 2016, pp. 4.7.1–4.7.4.
[5] S. Lee *et al.*, "Multi-level switching of triple-layered TaOx RRAM with excellent reliability for storage class memory," in *VLSIT*, 2012, pp. 71–72.
[6] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.
[7] B. Liu *et al.*, "Vortex: Variation-aware training for memristor X-bar," in *DAC*, 2015, pp. 1–6.
[8] Y. Zhu *et al.*, "Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise," in *DATE*, 2020, pp. 1590–1593.
[9] Y. Long *et al.*, "Design of reliable DNN accelerator with un-reliable ReRAM," in *DATE*, 2019, pp. 1769–1774.
[10] L. Chen *et al.*, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *DATE*, 2017, pp. 19–24.
[11] Z. Song *et al.*, "ITT-RNA: Imperfection tolerable training for RRAM-crossbar based deep neural-network accelerator," *TCAD*, pp. 1–14, 2020.
[12] C. Ma *et al.*, "Go unary: A novel synapse coding and mapping scheme for reliable ReRAM-based neuromorphic computing," in *DATE*, 2020, pp. 1432–1437.
[13] F. Zhang and M. Hu, "Defects mitigation in resistive crossbars for analog vector matrix multiplication," in *ASP-DAC*, 2020, pp. 187–192.
[14] S. Yu *et al.*, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, no. 10, p. 103514, 2011.
[15] F. Su *et al.*, "A 462GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory," in *VLSIT*, 2017, pp. C260–C261.
[16] M. Lin *et al.*, "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *ICCAD*, 2018, pp. 1–8.
[17] D. E. Rumelhart *et al.*, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
[18] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Elec. Comput.*, vol. EC-13, no. 1, pp. 14–17, 1964.
[19] "Nangate 45nm open cell library," 2008. [Online]. Available: http://www.nangate.com/