Write or Not: Programming Scheme Optimization for RRAM-based Neuromorphic Computing

Ziqi Meng[1], Yanan Sun[2], and Weikang Qian[1,3]
[1]UM-SJTU Joint Inst., [2]Dept. of Micro/Nano Electronics, and [3]MoE Key Lab of AI, Shanghai Jiao Tong Univ., China
{mengzq,sunyanan,qianwk}@sjtu.edu.cn

ABSTRACT

One main fault-tolerant method for a neural network accelerator based on resistive random access memory crossbars is the programming-based method, which is also known as write-and-verify (W-V). In the basic W-V scheme, all devices in crossbars are programmed repeatedly until they are close enough to their targets, which costs huge overhead. To reduce the cost, we optimize the W-V scheme by proposing a probabilistic termination criterion on a single device and a systematic optimization method on multiple devices. Furthermore, we propose a joint algorithm that assists the novel W-V scheme by incremental retraining, which further reduces the W-V cost. Compared to the basic W-V scheme, our proposed method improves the accuracy by 0.23% for ResNet18 on CIFAR10 with only 9.7% W-V cost under variation with $\sigma = 1.2$.

KEYWORDS

RRAM, reliability, variation, write-and-verify, neural network

1 INTRODUCTION

Resistive random access memory (RRAM) has a great potential for wide applications due to its non-volatility and high speed [1]. One application is to build RRAM crossbars to support fast in-situ vector-matrix multiplications (VMMs), which can provide a high-performance solution to neural network (NN) accelerators [2, 3].

However, an RRAM device suffers from resistance variation, which results in a deviation between the target value and the real one, and further leads to an accuracy drop in the NN accelerator. There are two different types of variation: device-to-device variation (DDV) and cycle-to-cycle variation (CCV). DDV causes devices at different locations to have different deviations. It is predictable before programming. CCV changes the resistance of the same device after another programming pulse, which is unpredictable [4].

Some methods are proposed to alleviate the impact of variation. They can be divided into the following categories.

Methods with error correction modules: These methods introduce error correction modules, such as redundant RRAM [5] and SRAM [6, 7], into the architecture to compensate the variation, at a cost of some hardware overhead.

(Re-)Training-based methods: These methods (re-)train NNs to enhance their robustness to device defects by modifying the objective function [8, 9], injecting disturbances in training [6, 10], or retraining the unmapped layers to recover the accuracy drop [11]. However, for a complex dataset or a large variation, they cost much runtime and even may not converge to a high accuracy.

Mapping-based methods: These methods map weights to appropriate devices to minimize the deviations caused by variation. After reorganization, the mapping order may be exchanged row by row [5, 8] or device by device [12]. They consume more routing or multiplexing cost for disordered inputs or require more devices to represent a weight. Besides, they can hardly handle CCV.

Programming-based methods: These methods repeatedly program a device followed by a read of the programmed value until it is close enough to the target value [13, 14]. They are also known as *write-and-verify (W-V)*. With limited endurance of RRAM devices [1], there should be an upper bound for the number of programming pulses applied to each device to prolong the chip lifetime.

The W-V method is one of the most common approaches as it tackles both DDV and CCV. Moreover, it can be combined with other methods to further boost their effectiveness. Thus, it is the focus of our work. In the basic version, as in [15], it applies repeated pulses on each device until the programmed value falls into a *target range* or an upper bound on programming times is reached. Despite the excellent fault-tolerant capability, it still has the following issues.

- *A larger final deviation:* With an upper bound on the programming times for a device, it is possible that the final programmed value does not fall into the target range and even has a larger deviation than some previously programmed values due to CCV (see Section 3.1 for more details).
- *Huge W-V cost:* In the basic scheme, each device may be programmed repeatedly until the upper bound is met. Besides, it reprograms all the devices in an NN accelerator, which may include millions of weights. In summary, it has a huge cost.

In this work, we propose optimization methods for the W-V process to address the above issues. The main contributions are as follows.

- For the W-V process on a single device, we propose a probabilistic early termination criterion to make the final programmed value get closer to the target in expectation, which reduces both the deviation and the W-V cost (see Section 3.2).
- For the W-V process on multiple devices, we formulate an optimization problem to reprogram only a subset of them with proper targets, to reduce the W-V cost (see Section 3.3). Notably, the target may differ from the original one in the proposed method. A quick but good enough solution is proposed (see Section 3.4).

*This work is supported by the National Key R&D Program of China under Grant No. 2020YFB2205501, National Natural Science Foundation of China under Grant No. 62174110, and Science and Technology Commission of Shanghai Municipality under Grant No. 20501130400. Corresponding author: Weikang Qian.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530558

- We further propose a joint algorithm (JA), where the novel W-V scheme is assisted by an incremental retraining (IR) technique. The JA tolerates both DDV and CCV, and can flexibly balance the retraining and W-V cost in different situations (see Section 4).

In experiments, the novel W-V scheme achieves higher accuracies at the same cost of the basic one. Compared to the basic W-V scheme, the JA improves the accuracy by 0.23% for ResNet18 on CIFAR10 with only 9.7% W-V cost under variation with $\sigma = 1.2$.

2 BACKGROUND

2.1 RRAM Crossbar for VMM

The basic operation in NNs is VMM in the form of $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{w}$, where $\boldsymbol{y}$ and $\boldsymbol{x}$ are vectors and $\boldsymbol{w}$ is a matrix. It can be efficiently implemented by a one transistor-one resistor (1T1R)-based RRAM crossbar, as shown in Fig. 1. The crossbar has vertical source lines (SLs) and horizontal bit lines (BLs) and word lines (WLs). There is a transistor and an RRAM device at each cross point. The conductance of the device at row i and column j encodes the element $w_{i,j}$ in the weight matrix $\boldsymbol{w}$. The voltages, which encode the input vector $\boldsymbol{x}$, are applied to the corresponding BLs. When all the WLs are activated, the current in the j-th SL is $y_j = \sum_i x_i w_{i,j}$ according to the Kirchhoff's law. Thus, the currents give the output vector $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{w}$.

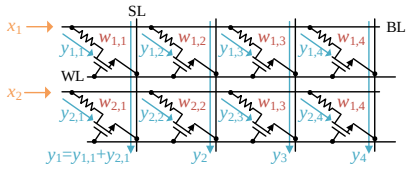

Figure 1: The VMM in the form of $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{w}$ performed by an RRAM crossbar.

However, it is hard to program an RRAM device to an arbitrary value $w_{i,j}$. Instead, an RRAM device can only switch among several states. A device with just two states, a *high resistance state (HRS)* and a *low resistance state (LRS)*, is called a *single-level cell (SLC)*. An SLC in HRS, which has low conductance, encodes the value 0, while one in LRS encodes 1. *Multi-level cells (MLCs)* have more conductance states, so they have higher encoding efficiency than SLCs [16]. Generally, multiple devices (either SLCs or MLCs) are used to represent a weight, where each device encodes some bit(s) of the weight. For simplicity, we will use SLCs as examples in the following methodology elaboration, which can be easily generalized to MLCs. In our experiments, MLCs are adopted.

In this work, we adopt a popular crossbar architecture for NN accelerators using two crossbars to encode a weight matrix, where one crossbar encodes the positive weights, the other encodes the negative ones, and their difference encodes the final results [3].

2.2 W-V Methods

An RRAM device suffers from variation. One common method to conquer variation is the W-V method, which repeatedly programs a device until the real value falls in a target range.

The W-V method has been well optimized at the circuit level, suitable for all RRAM-based applications. According to the type of the programming pulses, the W-V methods can be divided into the fixed-pulse method, the incremental-voltage method, and the incremental-pulse-width method [13, 14]. In the fixed-pulse method, pulses with the same magnitude and width are applied repeatedly. In contrast, the magnitude and the pulse width gradually increase in the incremental-voltage method and the incremental-pulse-width method, respectively. Besides, some advanced schemes adaptively adjust the programming pulses by feedback [13, 14].

The W-V methods are also refined for the application of NN accelerators. A parallel W-V scheme reduces the W-V time cost by programming multiple rows simultaneously [17]. Another work makes the W-V scheme more adaptive to the on-device training [18]. In comparison, we try to minimize the total W-V cost, including both the time cost and the energy consumption, under an acceptable accuracy drop. It is orthogonal to the other methods.

3 STATISTICAL OPTIMIZATION FOR W-V

This section presents our proposed optimization methods for the W-V process of an RRAM-based NN accelerator.

3.1 Motivation and Overview

To alleviate the problems mentioned in Section 1, we first focus on the W-V process for a single device given its target value. With limited device endurance [1], an upper bound on the programming times should be applied to each device, in order to maximize the device lifetime. However, due to the unpredictable CCV, it may not be beneficial to exhaust the maximally allowed programming times. For example, a device is expected to be programmed to a range $[0.9, 1.1]$ within 20 times, as shown in Fig. 2. Unfortunately, the programmed value is still out of the range after 19 programmings, although it is close to the range now. It is more likely to introduce a larger deviation by the 20th programming than the current one. As a result, we should stop the W-V process early to maintain a small deviation even if the upper bound has not been reached. Besides, early termination has another advantage that it leads to less W-V cost. Therefore, the first question is *when to stop programming a device within a limited number of programming times.*

Figure 2: Programmed values in a device with a target range $[0.9, 1.1]$ and an upper bound of 20 on programming times.

Figure 3: An example of the bit compensation within a weight, whose target is $12 = (1100)_2$. (a) Before W-V. (b) After two devices reprogrammed.

Besides, for a weight encoded by multiple devices, it is unnecessary to reprogram all of them because their deviations may cancel each other. For example, Fig. 3 shows 4 SLCs supposed to encode a target weight of $12 = (1100)_2$. The first programming for the four devices results in an inaccurate real weight of 14.5, as shown in Fig. 3(a). In the basic W-V scheme, all the four devices will be reprogrammed to their *original* targets, respectively. However, we can also obtain the target weight by only reprogramming two devices, i.e., the ones at the most significant bit (MSB) and the least significant bit (LSB), both to the target of 1, as shown in Fig. 3(b). Note that the original target for the LSB device is 0, but it is reprogrammed to 1 now. Thus, *each device can be changed to any state in the W-V process to minimize the total deviation of the weight, regardless of the original target.* As the total W-V cost is proportional

to the number of reprogrammed devices, the second question arises: *how to minimize the total deviation by reprogramming a minimized number of devices with proper targets.*

In this work, we provide answers to the above two questions. For the first question, we propose a *probabilistic early termination criterion* for the W-V process on a single device in Section 3.2. For the second question, we establish a 0-1 integer programming for the W-V process on multiple devices in Section 3.3, which is solved in Section 3.4.

3.2 Probabilistic Early Termination for W-V on a Single Device

In this section, we optimize the W-V process on a single device given its target and an upper bound on the programming times. Due to CCV, one more programming may not reduce the deviation but definitely incurs more W-V cost. Thus, it is unnecessary to reach the upper bound even if the programmed value has not fallen into the target range. We provide a probabilistic model to determine when to stop the W-V process. We first define some notations.

- The target for the device is $b \in \{0, 1\}$, while the real programmed value under variation is $\tilde{b}$. Both are known.
- The upper bound on the programming times for the device is t, which is a constant. Under t, the device can still be programmed for t' times at most when it has been programmed for $(t - t')$ times. For the example in Fig. 2, we have $t = 20$ and $t' = 1$.
- The programmed value by the k-th ($k \in \{1, 2, \ldots, t'\}$) programming pulse in future will be B^k, which is a random variable depending on b and k under variation. We assume that we can obtain its distribution by device characterization.

The basic W-V scheme on a device terminates if either 1) its real value falls into the target range, or 2) it has been programmed for t times. In our proposed method, we *include the above two criteria, plus a new criterion*, that is, the programmed value in future has a larger deviation than the current one. However, this is a random event, denoted as Y. We will stop the W-V process early if the probability of Y is more than a threshold P_{th}. In other words, the W-V process will be terminated early if the programmed value in future is more likely to have a larger deviation than the current one. In our work, we choose $P_{th} = 0.5$.

Next, we calculate the probability of event Y. Note that the probability of Y depends on $b, \tilde{b}$, and t'. Thus, we denote it as a conditional probability $P(Y|b, \tilde{b}, t')$. By definition, event Y happens if and only if $|B^k - b| > |\tilde{b} - b|$, for all $k \in \{1, \ldots, t'\}$. Thus, we have

$$P\left(Y|b, \tilde{b}, t'\right) = P\left(\bigcap_{k=1}^{t'} \left(\left|B^k - b\right| > \left|\tilde{b} - b\right|\right)\right).$$

Clearly, $P(Y|b, \tilde{b}, t')$ decreases with $|\tilde{b} - b|$. Let Δ^* satisfy that

$$P\left(\bigcap_{k=1}^{t'} \left(\left|B^k - b\right| > \Delta^*\right)\right) = P_{th}. \tag{1}$$

Thus, in order to check whether $P(Y|b, \tilde{b}, t') > P_{th}$, we only need to check whether $|\tilde{b} - b| < \Delta^*$.

Next, we derive Δ^*. We will show how to obtain it by taking the fixed-pulse W-V method as an example. Under this method, we can treat B^k's as independently and identically distributed, which means B^k is independent of k. Thus, we have

$$P\left(\bigcap_{k=1}^{t'} \left(\left|B^k - b\right| > \Delta^*\right)\right) = \left(P\left(\left|B^1 - b\right| > \Delta^*\right)\right)^{t'}. \tag{2}$$

By Eqs. (1) and (2), we further have

$$P\left(\left|B^1 - b\right| > \Delta^*\right) = \sqrt[t']{P_{th}}. \tag{3}$$

Note that the distribution of B^1 and the values of b, t', and P_{th} are known. Thus, Δ^* can be solved either analytically for a simple distribution of B^1 or numerically otherwise, based on Eq. (3).

At the system level, we need to do the termination check $|\tilde{b} - b| < \Delta^*$ for many devices after each programming. To reduce the repeated computation of Δ^* for each check, in our implementation, we pre-calculate the values Δ^* for each pairs of b and t', where $b \in \{0, 1\}$ and $t' \in \{1, \ldots, t\}$, and store them in a table indexed by b and t'. Then, for each device with b, t', and $\tilde{b}$ known, we can easily look up the table to check whether $|\tilde{b} - b| < \Delta^*$ to decide whether the W-V process should be terminated.

3.3 Systematic Optimization for W-V on Multiple Devices

In this section, we optimize the W-V process at the system level. Given a set of devices, we try to find a subset of them for reprogramming together with their new targets, in order to minimize both the total deviation and the W-V cost. Note that this optimization provides a plan before the W-V process starts. Then, the W-V scheme proposed in Section 3.2 is applied to each device to be reprogrammed under the guidance of this plan. Some new notations used in this section are as follows.

- The i-th target weight in the given set is w_i, which is an n-bit value. Its j-th ($1 \le j \le n$) bit is $b_{i,j} \in \{0, 1\}$. We have $w_i = \sum_{j=1}^{n} 2^{j-1} b_{i,j}$.
- The real value of the j-th bit in the i-th weight under variation is $\tilde{b}_{i,j}$, resulting in a real weight $\tilde{w}_i = \sum_{j=1}^{n} 2^{j-1} \tilde{b}_{i,j}$.
- In the W-V process, a new target for the j-th bit in the i-th weight is $b^*_{i,j} \in \{0, 1\}$, which may differ from $b_{i,j}$.
- In our work, only a subset of devices will be reprogrammed. The j-th bit in the i-th weight will (resp. will not) be reprogrammed if the indicator $d_{i,j} = 1$ (resp. $d_{i,j} = 0$).
- The values of $d_{i,1}, \ldots, d_{i,n}$ and $b^*_{i,1}, \ldots, b^*_{i,n}$ give a *reprogramming plan* for weight i. For simplicity, we denote it as $(d_{i,:}, b^*_{i,:})$.
- The final value of bit j in weight i after the W-V scheme in Section 3.2 is applied is $B^*_{i,j}$. It is a random variable on $b^*_{i,j}$ and t under variation.[1] We assume that $B^*_{i,j}$ follows a distribution $V(b^*_{i,j}, t)$, i.e., $B^*_{i,j} \sim V(b^*_{i,j}, t)$. Its expectation is denoted as $E[B^*_{i,j}]$.

Now, we formulate an optimization problem for the W-V process on the given set of devices as below:

$$\max_{d, b^*} \sum_i R_i(d_{i,:}, b^*_{i,:}) \tag{4}$$

$$\text{s.t. } R_i(d_{i,:}, b^*_{i,:}) = |w_i - \tilde{w}_i| - \left|w_i - W_i(d_{i,:}, b^*_{i,:})\right|, \tag{5}$$

$$W_i(d_{i,:}, b^*_{i,:}) = \sum_{j=1}^{n} 2^{j-1} \left[d_{i,j} E[B^*_{i,j}] + (1 - d_{i,j}) \tilde{b}_{i,j}\right], \tag{6}$$

$$\sum_i \sum_j d_{i,j} \le m,\ d_{i,j} \in \{0, 1\},\ b^*_{i,j} \in \{0, 1\},\ B^*_{i,j} \sim V(b^*_{i,j}, t). \tag{7}$$

By Eq. (6), $W_i(d_{i,:}, b^*_{i,:})$ is the *expected real weight (ERW)* under the reprogramming plan $(d_{i,:}, b^*_{i,:})$ for weight i. Thus, the term

[1] Note that the optimization problem will be solved before the W-V process starts, so all the devices can still be programmed for t times.

$|w_i - W_i(d_{i,:}, b^*_{i,:})|$ in Eq. (5) is the *expected deviation* of the weight under the reprogramming plan. The term $|w_i - \tilde{w}_i|$ in Eq. (5) is the current deviation of the weight, which is a known constant. Thus, by Eq. (5), $R_i(d_{i,:}, b^*_{i,:})$ is the *expected deviation reduction (EDR)* of weight i under the reprogramming plan. The objective is to maximize the total EDR over all the weights, as shown in Eq. (4), which is equivalent to minimizing the total expected deviation of the weights. To limit the total W-V cost, the number of reprogrammed devices is limited to m, as shown in Eq. (7). The unknowns are $d_{i,j}$'s and $b^*_{i,j}$'s. By solving them, we get an optimized W-V plan on which devices to reprogram as well as their reprogramming targets.

The above formulation needs the values $E[B^*_{i,j}]$. However, it is hard to obtain $E[B^*_{i,j}]$ analytically for the proposed the W-V process described in Section 3.2. Instead, we obtain it by the Monte Carlo simulation. Specifically, we repeatedly simulate the W-V process for target 0 and 1, respectively, where the proposed termination criteria are applied. The average of the final programmed values provides a good estimate for $E[B^*_{i,j}]$.

3.4 Heuristic Solution for the Systematic Optimization

It is challenging to find the optimal solution for the formulation shown in Eqs. (4)–(7). Instead, we present a greedy algorithm (GA) to solve it heuristically in this section. Its basic idea is to apply a sequence of reprogramming plans to the weights. To reduce complexity, each plan only reprograms one bit in a weight. Once a bit has been reprogrammed, we can treat that its corresponding indicator $d_{i,j}$ has been set to 1, so it will not be chosen again in the future. To elaborate the GA, we first introduce some definitions.

- *Single-bit reprogramming plan (SRP) of a weight*: a reprogramming plan of the weight such that only a single bit is reprogrammed. For weight i, assume the SRP reprograms its bit j ($1 \le j \le n$) with the target as $h \in \{0, 1\}$. We denote the SRP as $(j, h)_i$.
- *Promising plan (PP) of a weight:* an SRP $(j, h)_i$ of the weight such that 1) bit j has not been reprogrammed and 2) the EDR of the weight under the reprogramming plan is positive.
- *Most promising plan (MPP) of a weight*: if there exist PPs for the weight, the MPP is the one giving the maximal EDR; otherwise, it is *null*, meaning that the MPP does not exist.

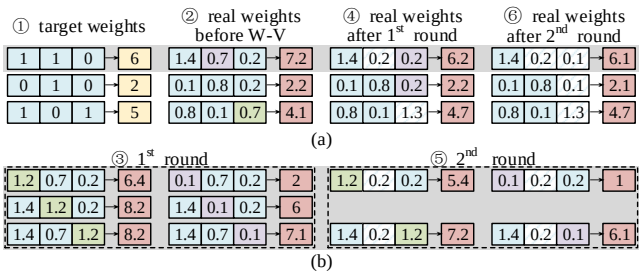

Figure 4: Illustrations for (a) the greedy algorithm and (b) its details for the first weight.

We use Fig. 4 to illustrate the above definitions. The target for weight 1 is $w_1 = 6$ (see ①). The real weight $\tilde{w}_1$ is 7.2 under variation (see ②). Assume that for all $1 \le i \le 3$ and $1 \le j \le 3$, we have $E[B^*_{i,j}] = 1.2$ (resp. 0.1) with target $b^*_{i,j} = 1$ (resp. 0). Weight 1 has 6 SRPs in total, which are $(j, h)_1$'s with $1 \le j \le 3$ and $h \in \{0, 1\}$. Their ERWs can be obtained by setting the corresponding bit to 1.2 or 0.1 (see ③). For example, the ERW of the SRP $(3, 1)_1$ is 6.4. By Eq. (5), its EDR is $|6 - 7.2| - |6 - 6.4| = 0.8$, which is positive. Thus, together with the fact that no bit has been reprogrammed, SRP $(3, 1)_1$ is a PP. In contrast, SRP $(2, 1)_1$ is not a PP since its EDR is $|6 - 7.2| - |6 - 8.2| < 0$. Among all the PPs, the SRP $(2, 0)_1$ gives the maximal EDR of 1.2. Thus, it is the MPP. Similarly, we can obtain that the MPPs of weights 2 and 3 are $(1, 0)_2$ and $(1, 1)_3$, respectively. Their EDRs are 0.1 and 0.5, respectively.

Our proposed GA is a loop. In each round, it checks all the weights and identifies the MPP of each weight. Among those weights with MPPs, it picks (at most) top $\lfloor m/n \rfloor$ weights with the largest EDRs given by the MPPs. Then, for each selected weight, the GA applies its MPP. When no MPP exists for any weight, the GA loop terminates. We select $\lfloor m/n \rfloor$ weights for reprogramming in each round because each selected weight in one round is reprogrammed for one bit. To ensure that each bit of a weight has a chance to be reprogrammed, we require the total number of rounds to be at least n. Meanwhile, we also want to reprogram as many bits as possible in each round. These considerations lead to the choice of $\lfloor m/n \rfloor$.

Now, we show an example on how our GA works using Fig. 4. In this example, $n = 3$. Assuming that $m = 6$, we have $\lfloor 6/3 \rfloor = 2$. In round 1, by the discussion above, all three weights have MPPs. The top 2 weights with the largest EDRs given by the MPPs are weights 1 and 3. Their MPPs are $(2, 0)_1$ and $(1, 1)_3$. Thus, bit 2 of weight 1 is reprogrammed with its target as 0 and bit 1 of weight 3 is reprogrammed with its target as 1. Assume that they are actually reprogrammed to 0.2 and 1.3 by the W-V scheme proposed in Section 3.2, respectively (see ④). Since the two devices have been reprogrammed, they will not be chosen again in any future rounds, indicated by coloring them in white. Before round 2 starts, $\tilde{b}_{1,2}$ and $\tilde{b}_{3,1}$ in the optimization problem should be replaced by 0.2 and 1.3, respectively. In round 2, by repeating the above steps, we find that the top 2 weights are weights 1 and 2. Their MPPs are $(1, 0)_1$ and $(1, 0)_2$, respectively (see the two purple squares in ④). Note that ⑤ in Fig. 4 shows the details on weight 1 in round 2: it shows the ERWs of the 4 remaining SRPs of weight 1. Assume that with the MPPs applied, bit 1 of weight 1 and bit 1 of weight 2 are actually both reprogrammed to 0.1 (see ⑥). In round 3, no weight has an MPP, since no one has a PP. Thus, the GA ends with a total deviation of 0.5.

The overall W-V scheme combines the optimized W-V plan obtained in this section and the W-V process proposed in Section 3.2. The overall flowchart is presented in the green box in Fig. 5. In each round, it first gets top k^* weights with their MPPs by the GA (see ⑥ in Fig. 5), where k^* equals the number of weights having MPPs if the number is less than $\lfloor m/n \rfloor$ or $\lfloor m/n \rfloor$ otherwise. Then, it repeatedly programs the bits specified in the MPPs based on the proposed termination criterion (see ⑧) and gets the final programmed values. Finally, it updates Eqs. (4)–(7) by replacing $\tilde{b}_{i,j}$'s with the new ones (see ⑨). These steps are repeated until no MPP exists for any weight (see ⑦) or the number of reprogrammed devices may exceed m (see ⑩). For the former case, the flag *noMPP* is set to *true*.

4 JOINT ALGORITHM ON W-V SCHEME

Besides the W-V process optimization, the inherent tolerance of NN provides a further chance to save the W-V cost. In this section, we will show how to remedy the impact of variation by the IR and then, how to combine the IR and the W-V process together.

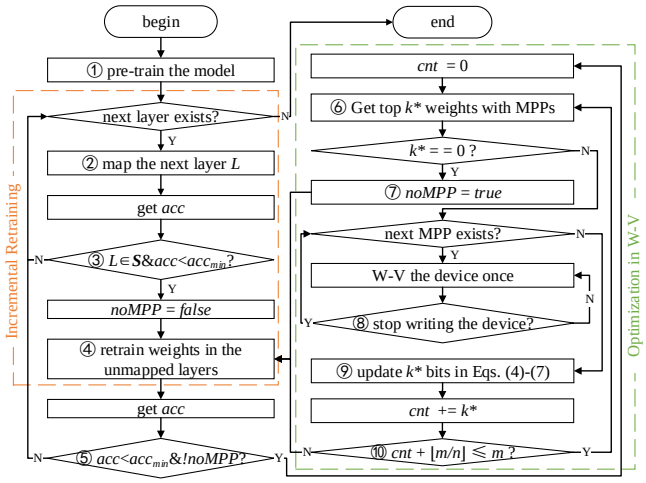

Figure 5: The joint algorithm.

4.1 Incremental Retraining

Although the accurate conductance value is hard to predict in advance, it can be easily read out after programming. Given the mapped weights, similar to the idea in [11], the remaining unmapped weights can be retrained to recover the accuracy. Note that we adopt the off-device training method here, which retrains the model only by software and has no impact to devices.

The IR process is shown in the orange box in Fig. 5. In each iteration, it first maps a layer L (see ②) and then gets the accuracy using the real weights of the already mapped layers and the ideal weights of the unmapped layers. To improve the accuracy, it retrains the unmapped weights (see ④) if the accuracy is lower than a threshold acc_{min} and L belongs to a subset $\boldsymbol{S}$ of all layers (i.e., $L \in \boldsymbol{S}$ in ③). For example, $\boldsymbol{S}$ only includes 5 layers out of a 20-layer NN. Note that when only the IR is applied, the conditional statement ⑤ will not be triggered and the IR continues to handle the next layer. However, IR alone may not deal well with large variation or complex datasets. Thus, it works as an assistance to the W-V method in our work.

4.2 Joint Algorithm

The proposed JA combines the optimized W-V scheme (Section 3) and the IR (Section 4.1) together, as shown in Fig. 5. First, it recovers the accuracy by the IR. If it fails and there exist MPPs (see ⑤), the layers just mapped after the last IR will be reprogrammed.

The two techniques in the JA tolerate both DDV and CCV. Furthermore, by tuning the parameters such as $\boldsymbol{S}$ and m, JA offers flexibility to trade between the retraining cost and the W-V cost. This can be exploited for different situations. For example, for a simple NN on a complex dataset, the W-V cost is relatively small compared to the retraining cost. Thus, we can tune the JA so that it overcomes the variation mainly by the W-V method with only a few retraining epochs. In contrast, for a deep NN on a simple dataset, the W-V cost is relatively larger due to a large number of weights. Thus, we can tune the JA so that it overcomes the variation mainly by the IR.

5 EXPERIMENTAL RESULTS

We compare six combinations of different techniques in this section, as listed in Table 1. We denote these six methods in italics in this section. Note that *IR* and *W-V* can provide an approximation for the methods in [11] and [15], respectively.

Table 1: Notation of different methods.

	w.o. W-V	basic W-V	proposed W-V
w.o. IR	*plain*	*W-V*	*W-V**
IR	*IR*	*W-V+IR*	*JA (W-V*+IR)*

We evaluate these methods from four aspects: accuracy, retraining cost, W-V cost, and the maximal programming times among all devices. The retraining cost is estimated by the number of training epochs and the W-V cost is estimated by the total programming times. The maximal programming times is an indicator for the impact of the W-V method to lifetime, the smaller the better.

5.1 Experimental Setup

Three benchmarks used in the simulations are ResNet18 with CIFAR10, ResNet18 with CIFAR100, and ResNet34 with CIFAR10, whose ideal accuracies are 94.14%, 74.36%, and 91.89%, respectively. During mapping, weights of each layer are first scaled to $[-255, 255]$ and then encoded by a pair of crossbars using 2-bit MLCs. The finite ON/OFF ratio is set to 200. Under log-normal variations, the relationship between the real value $\tilde{b}$ and the target b for each device is $\tilde{b} = be^{\theta}$, where θ is a normally distributed random variable with zero mean and standard deviation $\sigma \in [0.6, 1.2]$.

The fixed-pulse method is adopted in the W-V process, where the target range is set to $|b^* - \tilde{b}| < 0.1$. Unless otherwise specified, we reprogram $m = 20\%$ of all devices in each layer in *W-V** and *JA* except the last layer. The GA loops until no MPP exists in the last layer because this layer contributes more to the final accuracy. The upper bound for the programming times is set to $t = 20$ unless otherwise specified. To show the extreme situations, all devices are reprogrammed to the target range without an upper bound in *W-V*.

The accuracy threshold in the IR phase is set to $acc_{min} = 98\%$ on training set. In *JA*, ResNet18 can get retrained only after a set of layers $\boldsymbol{S} = \{L_5, L_9, L_{13}, L_{17}\}$, where L_i is the i-th layer in the NN. For a fair comparison to [11], $\boldsymbol{S}$ includes all layers in *IR*. The NN is retrained for 10 epochs in each round if not otherwise specified.

The simulation under each setting is repeated for 5 times with different initial $\tilde{b}$'s and the average result is reported. The results of *plain* are omitted in the following, where the accuracy of ResNet18 on CIFAR10 drops to 50% even under a small variation with $\sigma = 0.6$.

5.2 Accuracy

In this section, we compare the accuracy drops of ResNet18 on CIFAR10 for different methods. All of them work well under moderate variation with $\sigma < 1.0$, as shown in Fig. 6(a). Under large variation with $\sigma \geq 1.0$, however, accuracies of *IR* and *W-V** descend. To improve the accuracy, we can tune the hyper-parameters in the two methods. For example, the accuracy of *W-V** ascends to 92.78% by changing the upper bound t to 100.

Although *W-V* maintains a high accuracy, it costs huge W-V overhead. To illustrate the improvement by our proposed *W-V**, we compare the accuracy drops of the two methods under different W-V costs in Fig. 6(b). In *W-V*, the W-V cost depends on t, where all devices are reprogrammed. In *W-V**, we achieve different W-V costs

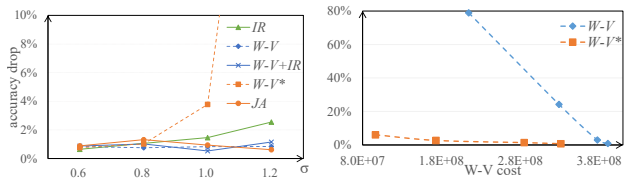

Figure 6: Accuracy drops of ResNet18 on CIFAR10 for different methods. (a) Accuracy drops under different σ's. (b) Accuracy drops with different W-V costs under $\sigma = 1.0$.

by tuning both m and t. In Fig. 6(b), *W-V** reaches an acceptable accuracy with a much smaller W-V cost than *W-V*.

5.3 Cost Estimation

In this section, we compare the W-V cost, the maximal programming times, and the retraining cost of the 6 methods. The normalized costs for ResNet18 on CIFAR10 with $\sigma = 1.2$ are shown in Fig. 7, where the costs of *JA* are set as the baseline (1, 1, 1). From Fig. 7, *JA* and *W-V** have much lower W-V cost than *W-V*. Specifically, *JA* just consumes 9.7% W-V cost even with an accuracy increase of 0.23% compared to *W-V*. Besides, *JA* and *W-V** reduce the maximal programming times drastically due to a small upper bound $t = 20$. Although *IR* only requires a little more retraining cost than *JA*, it has a lower accuracy as shown in Fig. 6(a).

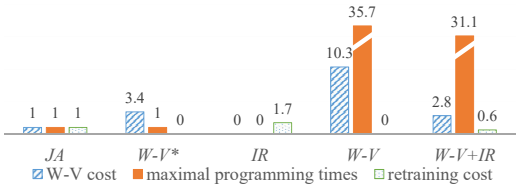

Figure 7: Normalized costs among different methods.

5.4 Trade-off in Joint Algorithm

As claimed in Section 4.2, the JA can make a trade-off between the retraining and the W-V cost for different situations. In this section, we demonstrate this with two benchmarks: ResNet34 on CIFAR10 and ResNet18 on CIFAR100. In the first benchmark, ResNet34 is much deeper and contains more weights, hence requiring more W-V cost if we apply the same parameters as those applied to ResNet18. On the other hand, the retraining process on simple CIFAR10 may converge fast. Thus, we mainly resort to the IR to conquer variation in this benchmark, such as by tuning retraining epochs and setting S to a larger one $\{L_3, L_7, L_{11}, L_{15}, L_{19}, L_{23}, L_{27}, L_{31}, L_{33}\}$. In contrast, CIFAR100 has more categories, so it is more difficult to train a high-accuracy NN on it. Thus, *JA* overcomes variation in the second benchmark mainly by the W-V method rather than the IR.

Fig. 8 shows the two types of costs for the two benchmarks, as well as their accuracy drops. Comparing points ② and ③, we find that to achieve an acceptable accuracy, the W-V cost is significantly reduced in ResNet34 by increasing a little retraining cost. In the second benchmark, however, the accuracy cannot reach high enough with IR alone even when the retraining cost is larger than that of the first benchmark, as shown by comparing points ① and ④. Instead, its accuracy ascends quickly when we reprogram more by increasing m and t, as shown by comparing points ④, ⑤, and ⑥. In conclusion, the JA can provide cost-effective solutions to different situations by allocating the two types of overhead reasonably.

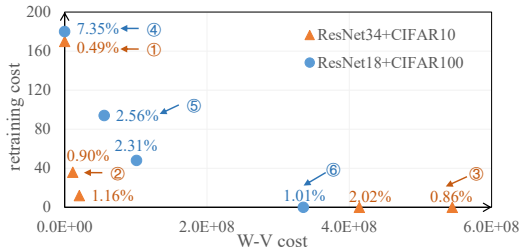

Figure 8: Trade-off between two types of overhead in different benchmarks, where labels are their accuracy drops.

Fig. 8 also shows better performance of our method than the existing methods. Compared to the method in [11], whose performance is indicated by points ① and ④, our method achieves an acceptable accuracy with less retraining cost, indicated by points ② and ⑤. For the method in [15], it requires much more W-V cost than the points ③ and ⑥ obtained by our method.

6 CONCLUSION

In this work, we propose a novel W-V scheme to reduce both the weight deviations and the W-V cost, including a probabilistic termination criterion on a single device and a systematic optimization on multiple devices. We also propose a joint algorithm that combines the novel W-V scheme and incremental retraining to provide a cost-effective solution by balancing the W-V and the retraining cost. The proposed method is orthogonal to many other methods.

REFERENCES

[1] S. Yu. *Resistive Random Access Memory (RRAM).* Morgan & Claypool, 2016.
[2] A. Shafiee et al. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, pages 14–26, 2016.
[3] P. Chi et al. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ISCA*, pages 27–39, 2016.
[4] A. Grossi et al. Fundamental variability limits of filament-based RRAM. In *IEDM*, pages 4.7.1–4.7.4, 2016.
[5] L. Chen et al. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *DATE*, pages 19–24, 2017.
[6] G. Charan et al. Accurate inference with inaccurate rram devices: Statistical data, model transfer, and on-line adaptation. In *DAC*, pages 1–6, 2020.
[7] Z. Meng et al. Digital offset for RRAM-based neuromorphic computing: A novel solution to conquer cycle-to-cycle variation. In *DATE*, pages 1078–1083, 2021.
[8] B. Liu et al. Vortex: Variation-aware training for memristor X-bar. In *DAC*, pages 1–6, 2015.
[9] Y. Zhu et al. Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise. In *DATE*, pages 1590–1593, 2020.
[10] Y. Long et al. Design of reliable DNN accelerator with un-reliable ReRAM. In *DATE*, pages 1769–1774, 2019.
[11] Y. Geng et al. An on-chip layer-wise training method for RRAM based computing-in-memory chips. In *DATE*, pages 248–251, 2021.
[12] C. Ma et al. Go unary: A novel synapse coding and mapping scheme for reliable ReRAM-based neuromorphic computing. In *DATE*, pages 1432–1437, 2020.
[13] K. Higuchi et al. 50nm hfo_2 ReRAM with 50-times endurance enhancement by set/reset turnback pulse & verify scheme. In *SSDM*, pages 1011–1012, 09 2011.
[14] J. K. Yoon et al. A 40nm 64Kb 56.67 TOPS/W read-disturb-tolerant compute-in-memory/digital RRAM macro with active-feedback-based read and in-situ write verification. In *ISSCC*, pages 404–406, 2021.
[15] P. Yao et al. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.
[16] S. Yu et al. Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory. *APL*, 98(10):103514, 2011.
[17] G. L. Zhang et al. An efficient programming framework for memristor-based neuromorphic computing. In *DATE*, pages 1068–1073, 2021.
[18] Y. Cai et al. Long live TIME: Improving lifetime for training-in-memory engines by structured gradient sparsification. In *DAC*, pages 1–6, 2018.