A Multifault-Tolerant Training Scheme for Nonideal Memristive Neural Networks

Yihong Chen, Zhen Fan, Shuai Dong, Minghui Qin, Min Zeng, Xubing Lu, Gougu Zhou, Xingsen Gao, and Jun-Ming Liu

Memristor crossbar is extensively investigated as an energy-efficient accelerator for neural network (NN) computations. However, hardware implementation of NNs using realistic memristors is challenging due to the ubiquity of faults (mainly classified into hard and soft faults) in memristors. Herein, a hardware-friendly, low-power multifault-tolerant training (MFTT) scheme capable of addressing both hard and soft faults simultaneously for memristive NNs is proposed. The MFTT scheme consists of multifault detection, targeted weight pruning, and in situ training with the Manhattan update rule. Specifically, multifault detection is first conducted to detect both hard and large soft faults. The detected faulty weights are subsequently pruned to prevent them from disturbing the NN. The sparsified NN after pruning is in situ trained so that the hard and large soft faults can be effectively tolerated using the sparsity and self-adaptivity of NNs. In addition, the remaining small soft faults can be well tolerated by the Manhattan update rule. Experimentally, MFTT demonstrates the lowest accuracy losses among several representative fault-tolerant schemes not only in the hard fault-only (when the hard fault ratio exceeds 10%) and soft fault-only (when the soft faults are large) cases, but also in the case where both types of faults coexist.

1. Introduction

Neural networks (NNs) have become superior in many artificial intelligence (AI) tasks—from simple tasks like image classification to complex tasks like autonomous driving.[1–4] State-of-the-art NNs contain an enormous number of weights and massive matrix-vector multiplications (MVMs). Implementation of these NNs on traditional computing platforms (e.g., central processing units (CPUs), graphic processing units (GPUs), and application specific integrated circuits (ASICs)) often results in high hardware cost and energy consumption, due to the memory wall of the von Neumann architecture.[5] Memristor crossbar has emerged as a computationally efficient and low-power solution to the NN hardware implementation.[6–8] By directly using Kirchhoff’s and Ohm’s laws, a memristor crossbar can perform the MVM operation in a parallel, analog fashion. Such capabilities of computing-in-memory and massive parallelism empower the memristor crossbar with high speed and energy efficiency for the NN processor application.[9,10]

Ideally, a memristor would have a linear and controllable response to a writing pulse, allowing its conductance (weight) to be precisely programmed to any target value. However, realistic memristors generally suffer from faults due to the immature fabrication and electroforming processes.[11] These faults are mainly classified into two categories: hard and soft faults. The hard faults include stuck-at-one (SA1) and stuck-at-zero (SA0) faults, denoting that memristors are stuck at the ON and OFF states, respectively. The soft faults include cycle-to-cycle (C2C) and device-to-device (D2D) variations, writing nonlinearity, finite number of conductance states, and limited ON/OFF ratio.[12–14] Memristors with hard faults are untunable, while those with soft faults can still be tuned but the actual programmed weights deviate from the target values. Therefore, both hard and soft faults can degrade the computational accuracies of memristor crossbar-based NNs significantly.
Tremendous research efforts have been devoted into improving the fault tolerance of the memristor crossbar-based NNs. To tolerate hard faults, there have been both hardware- and software-level solutions. The hardware-level solutions include switching off the individual access transistors connected to the faulty memristors\cite{16,17} and replacing the faulty memristors with redundant ones.\cite{18,19} These hardware-level solutions, however, introduce large area and routing overheads and complicate the design of peripheral circuits. On the other hand, the software-level solutions typically contain some or all of the following steps: adaptive weight mapping that maps the significant weights onto the fault-free memristors, weight pruning that fixes the weights mapped onto the faulty memristors to zero (or reducing these weights), and ex situ training (or retraining).\cite{18,19} The software-level solutions can compensate the hard faults-induced accuracy losses by taking advantage of the inherent sparsity of NN. In addition, they can save the hardware overhead. Therefore, the software-level solutions have become the mainstream in tolerating the hard faults.

For the tolerance of soft faults, the in situ training which can adjust the weights self-adaptively on chip is an effective approach.\cite{20} The fault tolerance can be further improved using some modified weight update rules, such as the Manhattan update rule,\cite{22,23} stochastic update rule,\cite{6,24} stochastic sparse update with momentum adaption,\cite{25} and sign-based backpropagation (BP) algorithm.\cite{26,27} The main idea of these modified weight update rules is to move the weights in the general direction that results in the reduction of loss function without regulating the step size.\cite{28} This makes the precise control over the weight update unnecessary and hence the soft faults can be tolerated.

As introduced earlier, the hard and soft faults have their respective solutions, but these solutions may not work properly when both faults are present simultaneously. For example, the mainstream solutions to hard faults typically use ex situ training, which is, however, difficult to load the software-computed weights to the memristor crossbar accurately in the presence of soft faults.\cite{11} On the other hand, the solutions to soft faults adopt in situ training, whose performance can be deteriorated by hard faults. This is because the in situ training attempts to update the weights mapped onto the memristors with SA0 and SA1 faults, while these weights are indeed untunable.\cite{20} Therefore, a general fault-tolerant solution capable of addressing both hard and soft faults simultaneously is urgently needed for the memristor implementation of NNs.

In this article, we propose a multifault-tolerant training (MFTT) scheme for rescuing the accuracies of memristive NNs with both hard and soft faults. This scheme includes three key steps: multifault detection, targeted weight pruning, and in situ training with the Manhattan update rule. The main contributions of this scheme are listed as follows. 1) Unlike previously proposed fault detection methods which mainly detected hard faults, a multifault detection method is proposed to detect both hard and large soft faults. 2) A weight represented by a pair of memristors is regarded as faulty if either one in the pair is detected to be faulty. Pruning is performed to the faulty weight by tuning the conductance of the normal memristor in the pair to be the same as (or close to) that of the faulty memristor. The pruned weight is thus zero (or close to zero), and it will no longer be updated during training. This process is called targeted weight pruning, which can effectively prevent the faulty weights from disturbing the NN. 3) The sparsified NN after pruning is in situ trained with the Manhattan update rule to recover the accuracy. Thanks to pruning, the accuracy losses caused by hard and large soft faults can be recuperated during the self-adaptive in situ training. Meanwhile, the remaining small soft faults can be well tolerated by a hardware-friendly Manhattan update rule.

The proposed MFTT scheme is applied to prototype single-layer perceptron (SLP) and multilayer perceptron (MLP) for modified national institute of standards and technology (MNIST) image recognition. In the hard faults-only case, MFTT achieves an accuracy loss of 2.7% at the hard fault rate of 20%, which is better than the performances of previous hard fault-tolerant schemes. In the soft fault-only case, MFTT outperforms previous in situ training-only schemes, particularly when the soft faults are large. Moreover, when both hard and soft faults are present simultaneously, MFTT exhibits the highest accuracy among all the investigated schemes.

2. Preliminaries

2.1. Memristor Crossbar-Based NN

Memristor is a two-terminal passive circuit element whose conductance can be continuously tuned by applied electrical pulses. The memristor conductance can be used to represent the synaptic weight due to its multivalue, nonvolatility, and tunability. An array of interconnected memristors form a memristor crossbar which can store a matrix of weights as conductances. Based on Ohm’s and Kirchhoff’s laws, when applying a vector of voltage pulses along the rows of a memristor crossbar, the currents collected along the columns are

\[
I_j = \sum_i G_{ij} \cdot V_i
\]  

where \(I_j\) is the output current along the \(j\)-th column, \(V_i\) is the input voltage along the \(i\)-th row, and \(G_{ij}\) is the memristor conductance at the cross point of \(i\)-th row and \(j\)-th column. Equation (1) indeed represents an MVM operation, and its time complexity is reduced to \(O(1)\) using the memristor crossbar.

Because \(G_{ij}\) is always positive, a pair of memristors\cite{22} or additional reference memristors\cite{13} are needed to represent a signed weight \(w_{ij}\). Here, two differential crossbars with memristor pairs are used, as shown in Figure 1a. The difference between \(G_{ij}^+\) and \(G_{ij}^-\) can thus represent a signed weight \(w_{ij}\).

Here, one op-amp circuit is assumed for one column, but for a large-scale crossbar, a multiplexer may be used to allow several columns to share one op-amp circuit. Using the op-amp circuits, the output currents are converted to voltages, which may be further fed to analog-to-digital converters (ADCs) to produce digital data. The digital data are stored and used for further computations (e.g., the activation function). However, the multiplexer, ADCs, and digital logic circuits as needed are not shown in Figure 1a, which can be referred to in other studies.\cite{15,29}
2.2. Hard and Soft Faults of Memristors

Hard faults include SA1 and SA0 faults, manifesting as those memristors that are stuck at the ON and OFF states, respectively. Typically, the SA1 fault results from the overforming defects while the SA0 fault stems from the open-switch defects. The SA1 and SA0 faults can be randomly distributed or clustered in a row (or column) in a memristor crossbar. Here, only the random distribution is considered. Because a pair of memristors are used to represent a signed weight, one or both of the memristors may be faulty. Table 1 shows the possible SA0–SA1 combinations in the differential crossbars when only hard faults are considered. C1 denotes the case where both positive and negative memristors are fault free. In this case, weights in the whole range of $[-w_{\text{max}}, +w_{\text{max}}]$ can be successfully mapped. C2–C5 denote the cases where one of the memristors has either SA0 or SA1 fault, while the other one is fault free. In these cases, weights in only half of the range of $[-w_{\text{max}}, +w_{\text{max}}]$, that is, $[-w_{\text{max}}, 0]$ or $[0, +w_{\text{max}}]$, can be mapped. C6–C9 denote the cases where both two memristors have either SA0 or SA1 fault. In these cases, weights are fixed at 0, $-w_{\text{max}}$, or $+w_{\text{max}}$.

Soft faults refer to those causing the memristor conductance to deviate from the ideal value (but the conductance can still be tuned). The soft faults include C2C and D2D variations, writing nonlinearity, finite number of conductance states, and limited ON/OFF ratio. For a realistic memristor, the conductance can be tuned only in a finite range of $[G_{\text{min}}, G_{\text{max}}]$ by applying writing pulses, and this range is quantified by the ON/OFF ratio ($G_{\text{max}}/G_{\text{min}}$). In addition, the number of conductance states ($N_s$) available in the range of $[G_{\text{min}}, G_{\text{max}}]$ is also a finite value. During writing, the conductance change depends nonlinearly on the number of writing pulses, which is referred to as writing nonlinearity ($\sigma$). Every writing pulse can introduce a noise to the conductance change. This fault is called the C2C variation ($\sigma_{\text{C2C}}$) and $\sigma_{\text{C2C}}$ typically follows Gaussian distribution. In addition, the conductance change induced by the same number of pulses can vary from device to device, a fault known as the D2D variation ($\sigma_{\text{D2D}}$). $\sigma_{\text{D2D}}$ is difficult to be quantified because the differences of $G_{\text{max}}/G_{\text{min}}$, $N_s$, $\sigma$, and $\sigma_{\text{C2C}}$ among different devices can all cause $\sigma_{\text{D2D}}$. The aforementioned soft faults are schematically shown in Figure 1b.

2.3. Previous Fault-Tolerant Schemes

Most previous fault-tolerant schemes addressed only one type of faults, either hard or soft faults. To tolerate the hard faults, both hardware- and software-level solutions have been investigated. Chen et al. proposed using the 1T1R array architecture for the memristor crossbar and hence the faulty memristors could be pruned by switching off the individual access transistors connected to them. Liu et al. utilized the redundant columns of memristors to replace the faulty columns onto which significant weights were mapped. Although the hardware-level solutions can alleviate the accuracy losses caused by hard defects, they introduce large-area and routing overheads and also increase the design complexity of peripheral circuits. To circumvent these issues, a variety of software-level solutions have been proposed. Liu et al. applied a fault-aware retraining scheme to SLP on the MNIST dataset (note: the MNIST dataset is always used hereafter unless otherwise specified), recovering the accuracy to 98.1% of the ideal value in the presence of 20% hard faults. Chen and Song et al. used a bipartite-matching algorithm to map significant weights to the fault-free memristors. They further reduced the large weights mapped onto the faulty memristors and then
performed retraining. Their scheme implemented on MLP could achieve $\leq 1\%$ loss of accuracy at a hard fault ratio of 20%. Xia et al.\[^{20}\] proposed a scheme combining online fault detection and fault-tolerant training. In the training phase, fault-blind weight pruning was performed in software, followed by genetic algorithm-based remapping. Their scheme could recover the accuracy of the VGG-11 network (on the Cifar-10 dataset) from 37% to 83% when 14% hard faults are present. Jin et al.\[^{18}\] modified Xia et al.’s scheme using targeted weight pruning followed by retraining of the sparsiﬁed network. Their scheme implemented on a four-layer convolutional NN (CNN) could reduce the accuracy loss to within 3% at a hard fault ratio of 20%. Although all the software-level solutions could effectively tolerate the hard faults, their performance may degrade signiﬁcantly when the soft faults are present simultaneously. This is because calculating and storing the intermediate data. For all these modiﬁed weight update rules) to address large soft faults, which were allowed by in situ training. To further improve the fault tolerance, various modiﬁed weight update rules have been studied. Lim et al.\[^{22}\] used the Manhattan update rule for in situ training, where the weight was increased or decreased by a single step according to the sign of the backpropagated error. The MLP trained with the Manhattan update rule achieved accuracies close to the ideal values when $\nu$ was smaller than 3 and $n_i$ was larger than 32. Gokmen et al.\[^{6}\] proposed a stochastic update rule which translated the inputs and errors to stochastic bit streams and reduced the input-error multiplication to a simple AND operation. The stochastic update rule resulted in an accuracy loss of 0.3% for an MLP at a noise of 10%. Zhang et al.\[^{27}\] applied a sign-based BP algorithm, as it has been shown that it can improve the fault tolerance. Jin et al.\[^{18}\] proposed to detect both hard and soft faults. Then, the targeted weight pruning is performed to force the detected faulty weights to be zero (or close to zero) based on the self-compensation mechanism. After that, the in situ training with the Manhattan update rule is implemented on the sparsiﬁed NN. The detailed methods of the multifault detection, targeted weight pruning, and in situ training are described in the next three subsections.

### 3. MFTT Scheme

#### 3.1. Overview

Figure 2 illustrates the overall ﬂow of the proposed MFTT scheme. The multifault detection is ﬁrst carried out to detect both hard and large soft faults. Then, the targeted weight pruning is performed to force the detected faulty weights to be zero (or close to zero) based on the self-compensation mechanism. After that, the in situ training with the Manhattan update rule is implemented on the sparsiﬁed NN. The detailed methods of the multifault detection, targeted weight pruning, and in situ training are described in the next three subsections.

#### 3.2. Multifault Detection

Some efﬁcient methods have been recently proposed to detect the types and locations of faults in memristor crossbars, but they mainly focused on hard faults, while soft faults were unaddressed.\[^{20,34,35}\] Song et al.\[^{19}\] proposed to detect the conductance variation (i.e., soft fault) of every memristor and record it together with the location of this memristor in a buffer. This bit-wise detection method is relatively slow and the stored information of every memristor’s conductance variation value is redundant.

Here, we propose a simple multifault detection method to detect the locations of both hard and large soft faults. The proposed multifault detection can be performed row by row for an individual crossbar. The connection between the two differential crossbars can be temporarily cut off using some switches in the circuit (not shown in Figure 1a). We first initialize the conductances of a row of memristors (e.g., the ﬁrst row) to $G_{\text{max}}$ by applying sufﬁcient numbers of potentiation pulses. If an ideal memristor has $n_i$ conductance states in the range of $[G_{\text{min}}, G_{\text{max}}]$, applying potentiation pulses with a number of $P_1 = (n_i - 1)$ is sufﬁcient to increase the conductance to $G_{\text{max}}$. Then, depression pulses with a number of $P_2$ (e.g., $P_2 = 16$ for $n_i = 32$) are applied along this row. For an ideal memristor, its conductance after depression would become

$$G_{\text{ideal}} = G_{\text{max}} - (G_{\text{max}} - G_{\text{min}}) \cdot \frac{P_2}{n_i - 1} \quad (2)$$

However, the conductance of an actual memristor ($G_{\text{actual}}$) can deviate from $G_{\text{ideal}}$ signiﬁcantly in the following cases: 1) it has either an SA0 or SA1 fault; 2) it has a writing nonlinearity $\nu$ or 3) a writing noise $\sigma_{\text{CIC}}$ much larger than zero; 4) it has an ON/OFF

### 2.4. Motivation

As introduced in Section 2.3, although the hard and soft faults have their respective solutions, these solutions may have limited performances when both faults are present simultaneously. This motivates us to develop a so-called MFTT scheme capable of addressing both hard and soft faults simultaneously. Our first consideration is that the inherent sparsity of NN should be taken full use of. We therefore propose to ﬁrst detect both hard and large soft faults using a multifault detection method and then prune the detected faulty weights. The sparsiﬁed NN after pruning can be in situ trained to recover the accuracy due to self-adaptivity. In addition, the small soft faults remained in the NN can be well tolerated using the Manhattan update rule for in situ training. Therefore, both hard and soft faults can be addressed by the proposed MFTT scheme, which combines the multifault detection, targeted weight pruning, and in situ training.
ratio \( G_{\text{max}}/G_{\text{min}} \) or 5) conductance state number \( n_s \), much smaller than their respective ideal values. Case (1) and (2–5) represent the hard and soft faults, respectively, and their schematic illustrations are shown in Figure 3a-g. For Case (4), because the memristors typically exhibit a small variation of \( G_{\text{max}} \) after electroforming with appropriate voltages and compliance currents,\[^{36}\] only the variation of \( G_{\text{min}} \) is considered to be responsible for the variation of \( G_{\text{max}}/G_{\text{min}} \).

To characterize the deviation between \( G_{\text{actual}} \) and \( G_{\text{ideal}} \), a reading pulse is applied along the first row of the individual crossbar and the output voltages along different columns are obtained in parallel as follows.

\[
V_{o,j} = G_{ij} \cdot V_R \cdot R_0 \tag{3}
\]

where \( V_{o,j} \) is the output voltage at the \( j \)-th column, \( G_{ij} \) is the actual conductance of the \( j \)-th memristor in the first row, \( V_R \) is the reading voltage, and \( R_0 \) is a constant resistance used in the op-amp circuit (see Figure 1a). Note that Equation (3) applies to the case where an individual crossbar is being detected, while its connection to the neighboring crossbar is temporarily cut off. In addition, because the op-amp circuit (shown in Figure 1a) outputs a voltage with the negative sign, another op-amp circuit is needed to invert the sign of the output voltage. The output voltage \( V_{o,j} \) is subsequently compared with a reference voltage \( V_f \) given by

\[
V_f = G_{\text{ideal}} \cdot V_R \cdot R_0 \tag{4}
\]

The difference between \( V_{o,j} \) and \( V_f \) can thus be used to screen out the memristors belonging to Case (1–5). Specifically, when \( |V_{o,j} - V_f| > \epsilon \) (\( \epsilon \) is a threshold value), the memristor shall have either a hard fault or a large soft fault, as described in Case (1–5); otherwise, it is free from the hard and large soft faults. This screening operation can be implemented using the circuit, as shown in Figure 3h. Note that this circuit may have limitations such as susceptibility to noise and difficulty to settle when the input is very close to \( \pm \epsilon \), which are not considered in this work.

Repeating the above four steps (initialization, writing, reading, and screening) for the rest rows, all the memristors with hard and large soft faults in a crossbar are thus detected. Their locations are stored in a buffer. The remaining memristors either have small soft faults or are fault free. Note that the proposed method can detect multifaults without identifying the exact fault types. In addition, the conductance variation values are not stored. Therefore, our method appears to be simpler and more efficient than the previous fault detection methods when dealing with multifaults.\[^{19,20}\]

### 3.3. Targeted Weight Pruning

After multifault detection, the targeted weight pruning is performed for the two differential crossbars based on the self-compensation mechanism. It has been demonstrated that more than 50% weights in an NN (particularly for a fully connected NN) could be pruned, while causing almost no accuracy loss in certain tasks.\[^{37}\] This motivates us to fix the faulty weight to be zero (if either one in a memristor pair is detected to be faulty, the corresponding weight is regarded as faulty), using the normal one in the pair to compensate the faulty one (i.e., self-compensation). Prior to performing it, a clear understanding of possible fault combinations in the memristor pairs is needed. When both hard...
and soft faults are present simultaneously, the fault combinations considered in previous studies\[^{18}\] (see Table 1) should be extended.

Table 2 shows the possible combination of hard faults (SA0 and SA1 faults), large soft faults, and “normal” devices in the differential crossbars. Here, “normal” denotes that the memristor is fault free or has a small soft fault. The normal memristors have been distinguished from those with hard and large soft faults using the earlier-described multifault detection. The multifault detection, however, does not further distinguish the memristors with SA0 and SA1 faults and large soft faults. Nevertheless, they are listed as independent cases in Table 2 because they will respond differently to the writing pulses applied in the pruning step (to be described later).

According to Table 2, one can use a pair of normal memristors (C1) to map the weights in the whole range of \([-w_{\text{max}}, +w_{\text{max}}]\). The NN training can thus fully rely on these memristors. In C2–C3 where both memristors have SA0 or SA1 faults, the weights are naturally pruned. In C4–C10 where at least one memristor has an SA1 or large soft fault, the weights cannot be mapped appropriately. These weights need to be pruned, which can be realized by writing the conductances of the paired memristors to \(G_{\text{max}}\). Likewise, in C11–C14, where at least one memristor has an SA0 fault, the weight pruning is also needed, which can be realized by writing the conductances of the paired memristors to \(G_{\text{min}}\). Note that in C4–C14, the weights after pruning may not be exactly zero, because the conductance of the memristor with a soft fault may not be tuned to be exactly the same as that of its neighboring memristor with hard or soft faults, but a weight fixed to a close-to-zero value is still called a pruned weight. In the last two cases (C15–C16), where one memristor has an SA0 fault, while the other has an SA1 fault, the weights are fixed to \(+w_{\text{max}}\) and \(-w_{\text{max}}\) and they cannot be pruned.

To implement the targeted weight pruning efficiently, a row-by-row manner is adopted. Specifically, taking the first row of two

---

**Figure 3.** Schematic illustrations of the different conductance changes of memristors having a) no fault, b) SA1 fault, c) SA0 fault, d) large \(v\), e) large \(\sigma_{\text{C}_2}\), f) small \(n_s\), and g) small \(G_{\text{max}}/G_{\text{min}}\) in response to depression pulses (the pulse number of 1 is used for a simple illustration). The conductances of all memristors (except SA0) are initialized to \(G_{\text{max}}\) beforehand. h) Circuit to implement the screening operation.
Table 2. Fault combinations and corresponding available weight ranges for the case where both hard and soft faults are present. LSF is an abbreviation for “large soft fault.”

| Case  | –Memristor | –Memristor | Weight can be mapped |
|-------|------------|------------|---------------------|
| C1    | Normal     | Normal     | \([-w_{\text{max}}, +w_{\text{max}}]\) |
| C2    | SA0        | SA0        | 0                   |
| C3    | SA1        | SA1        | 0                   |
| C4    | Normal     | SA1        | \([-w_{\text{max}}, 0]\) |
| C5    | SA1        | Normal     | \([0, +w_{\text{max}}]\) |
| C6    | LSF        | Normal     | \([-w_{\text{max}}, +w_{\text{max}}]\) |
| C7    | Normal     | LSF        | \([-w_{\text{max}}, +w_{\text{max}}]\) |
| C8    | LSF        | SA1        | \([-w_{\text{max}}, 0]\) |
| C9    | SA1        | LSF        | \([0, +w_{\text{max}}]\) |
| C10   | LSF        | LSF        | \([-w_{\text{max}}, +w_{\text{max}}]\) |
| C11   | Normal     | SA0        | \([0, +w_{\text{max}}]\) |
| C12   | SA0        | Normal     | \([0, +w_{\text{max}}]\) |
| C13   | LSF        | SA0        | \([0, +w_{\text{max}}]\) |
| C14   | SA0        | LSF        | \([-w_{\text{max}}, 0]\) |
| C15   | SA0        | SA1        | \(-w_{\text{max}}\) |
| C16   | SA1        | SA0        | \(+w_{\text{max}}\) |

differential crossbars as an example, a sufficient number of potentiation pulses \([P_1 = (n_s - 1)]\) are applied to the selected memristor pairs through the columns to set the conductances of these memristors to \(G_{\text{max}}\). The memristor pairs are selected if either device in the pair is detected to be faulty (C2–C16 in Table 2). This requires the use of control signals based on the stored fault locations.

Then, a reading pulse is applied along the first row of the two differential crossbars and the output voltages along different columns are obtained in parallel as follows.

\[
V'_{i,j} = (G_{ij}^+ - G_{ij}^-) \cdot V_R \cdot R_0 \tag{5}
\]

Because writing has been performed beforehand, \(G_{ij}^+\) and \(G_{ij}^-\) which are tunable are set to be around \(G_{\text{max}}\), while those corresponding to hard fault remain unchanged. Therefore, \(V'_{i,j}\) can be exactly zero or close to zero for the cases of C2–C5 in Table 2. In contrast, \(V'_{i,j}\) can deviate from zero significantly for the cases of C6–C16 in Table 2. C6–C16 can be further divided into two groups according to the degree of deviation: 1) C6–C10, where at least one memristor has a large soft fault while the other has no SA0 fault and 2) C11–C16, where one memristor has SA0 fault. Apparently, the deviation of \(V'_{i,j}\) from 0 (i.e., \(|V'_{i,j}|\)) should be larger in C11–C16 than in C6–C10.

To distinguish the above cases according to \(V'_{i,j}\), one can use a sufficiently large threshold value \(\epsilon'\) and a circuit similar to that shown in Figure 3h. \(|V'_{i,j}| \leq \epsilon'\) screens out the cases of C2–C10. The weights leading to \(|V'_{i,j}| \leq \epsilon'\) are regarded as already pruned, and no more operation will be performed on them. In contrast, \(|V'_{i,j}| > \epsilon'\) screens out the cases of C11–C16. The weights leading to \(|V'_{i,j}| > \epsilon'\) may be further adjusted because the memristors with soft faults are still tunable.

Rewriting is thus performed for the first row by applying a sufficient number of depression pulses \([P_1 = (n_s - 1)]\) to each column where \(|V'_{i,j}| > \epsilon'\) is detected. After rewriting, the weights in C11–C14 can be pruned while those in C15–C16 remain stuck.

Repeating the earlier four steps (writing, reading, screening, and rewriting) for the rest rows, the weights in the differential crossbars belonging to C2–C14 can all be pruned. These pruned weights will not be adjusted any more in the weight update step of the in situ training, as described as follows.

### 3.4. In Situ Training with Manhattan Update Rule

The in situ training of a memristor crossbar-based NN mainly includes four steps: forward propagation, backward propagation, gradient calculation, and weight update. Using memristor pairs \((G_{ij}^+ - G_{ij}^-)\) to represent signed weights \(w_{ij}\), the forward propagation is implemented by applying voltages encoded by the inputs \(x_i\) (or activations \(a_i^{(l-1)}\) of the \((l-1)\)-th layer) to the rows and obtaining the output currents representing the weighted sums \(s_j^{(l)}\) along the columns. Subsequently, \(s_j^{(l)}\) is fed to an activation function \(f\) and thus converts to \(a_j^{(l)}\) of the \(l\)-th layer. This forward propagation continues until the output layer (i.e., the \(L\)-th layer) is reached. The errors \(\delta_i^{(l)}\) between the actual outputs \(y_j^{(l)}\) and the target outputs \(t_j\) are computed and then used for backward propagation. In backward propagation, the voltages encoded by the errors of the \(l\)-th layer \(\delta_i^{(l)}\) are applied along the columns and the output currents representing the backward weighted sums \((\sum w_{ij}\delta_i^{(l)})\) are obtained along the rows.

Subsequently, the errors of the \((l-1)\)-th layer \(\delta_i^{(l-1)}\) are obtained by multiplying \(\sum w_{ij}\delta_i^{(l)}\) by the derivative values of the activation function \(f'(s_j^{(l-1)})\). After the errors of all the layers are obtained, the gradients of the loss function \(f\) can be calculated as

\[
\frac{\partial f}{\partial w_{ij}} = a_j^{(l)} \delta_i^{(l+1)} \tag{6}
\]

As seen earlier, the memristor crossbars are used to implement the MVM operations during both forward and backward propagations. Other operations involved are all hardware implementable using digital logic circuits.[29] Now, let us focus on the weight update. The Manhattan update rule is used for its ease of hardware implementation and its fault tolerance.[22,28] In the Manhattan update rule, weight is increased or decreased by only one step according to the sign of \(\partial f/\partial w_{ij}\), as expressed by

\[
w_{ij}^{(t+1)} := w_{ij}^{(t)} - \eta \cdot \text{sgn} \left( \frac{\partial f}{\partial w_{ij}} \right) \tag{7}
\]

where \(\eta\) is the minimum allowed weight update corresponding to the conductance change induced by only one potentiation/depression pulse. However, some weights should not be updated.
because they are already pruned. In addition, to avoid the oscillation issue and reduce the number of update events, there is no need to update the weight with a small magnitude of \( \partial J / \partial w_j \). We therefore use two mask matrices \( M_{1,ij} \) and \( M_{2,ij} \) whose elements are either 0 or 1 to regulate the weight update. In \( M_{1,ij} \), the element 0 (1) indicates that the corresponding weight is pruned (not pruned). In \( M_{2,ij} \), the element 0 (1) indicates that the corresponding \( |\partial J / \partial w_j| \) is smaller (larger) than a threshold value \( \epsilon_{\text{grad}} \). 

\( M_{1,ij} \) is fixed because its elements are generated from the stored fault locations, while \( M_{2,ij} \) can vary from iteration to iteration because its elements are generated from the intermediate results. By taking into account \( M_{1,ij} \) and \( M_{2,ij} \), Equation (7) is modified as

\[
w^{(l)}_{ij} = w^{(l)}_{ij} - M_{1,ij} \times M_{2,ij} \cdot \text{sgn} \left( \frac{\partial J}{\partial w_j} \right)
\]  

Equation (8)

The weight update based on Equation (8) can be performed row by row on the differential crossbars. Using the first row as an example, \( M_{1,1j} \) and \( M_{2,1j} \) are first used as the control signals to select the columns intended to be updated. Then, two-phase writing is performed, as schematically shown in Figure 4. The first phase deals with the case of \( \text{sgn}(\partial J / \partial w_{ij}) = -1 \). \( G^+_{ij} \) will be increased by one step if \( G^+_{ij} < G_{\text{max}} \), while \( G^-_{ij} \) will be decreased by one step if \( G^-_{ij} = G_{\text{max}} \). The second phase deals with the case of \( \text{sgn}(\partial J / \partial w_{ij}) = +1 \). \( G^-_{ij} \) will be increased by one step if \( G^-_{ij} < G_{\text{max}} \), while \( G^+_{ij} \) will be decreased by one step if \( G^+_{ij} = G_{\text{max}} \). Apparently, prior to the two-phase writing, a reading step is required to check whether \( G^+_{ij} \) and \( G^-_{ij} \) have reached \( G_{\text{max}} \). Repeating the above two steps of reading and two-phase writing for the rest rows, the weight update for one iteration is completed. The total number of iterations is set appropriately so that the convergence can be reached.

![Figure 4](https://www.advancedsciencenews.com/)

Figure 4. Schematics illustrating a) the column selection based on the mask matrices \( M_{1,ij} \) and \( M_{2,ij} \), and the b) first and c) second phases of writing for memristors with \( \text{sgn}(\partial J / \partial w_{ij}) = -1 \), and +1, respectively. Note that all the \( M_{1,ij} \), \( M_{2,ij} \), \( \text{sgn}(\partial J / \partial w_{ij}) \), and \( G_{ij} \) values are assumed for memristors for the convenience of illustration. \( V_p \) and \( V_d \) are the voltages of the potentiation and depression pulses, respectively, and the half-bias method is used.
4. Results and Discussion

4.1. Simulation Details

Table 3 lists the simulation details. The efficiency of the MFTT scheme is evaluated on an SLP (784 × 10) and an MLP (784 × 256 × 10) for image recognition. 60,000 and 10,000 images of handwritten digits from the MNIST dataset[38] are used for the training and test, respectively. The SLP is implemented using a pair of crossbars (each containing 7840 memristors) together with 784 input neurons and 10 output neurons (using the softmax function). On the other hand, MLP is implemented using a 200704-memristor crossbar pair and a 2560-memristor crossbar pair for the first and second layers, respectively, together with 784 input neurons, 256 hidden neurons (using the ReLu function), and ten output neurons (using the softmax function). The crossbar is assumed to be sufficiently large, and thus the tiled architecture is not used. Only the effects of the faults of memristors on the network performance are investigated, while other effects (such as ADC precision and sneak path) are not considered.

A fault-free memristor is assumed to have a writing nonlinearity (ψ) of 0, a number of conductance states (N) of 32, an ON/OFF ratio (Gmax/Gmin) of 10, a writing noise (σC2C) of 0, and a device-to-device variation (σDD) of 0. The soft faults are introduced by increasing ψ and σC2C and decreasing N and Gmax/Gmin. For Gmax/Gmin, only Gmin is varied while Gmax assumed to be unchanged, as mentioned earlier. To introduce σDD, memristors are assumed to be independent and each individual has a random set of μ, N, Gmax, Gmin, and σC2C. The μ, N, Gmax, Gmin, and σC2C values are assumed to obey the uniform distribution. For example, all the memristors have independent values, and these values are uniformly distributed in the range of [0, vmax]. vmax = 0 means that the memristors have an ideal value of 0 and no σDD exists. As vmax increases, both the average μ and σDD become larger. Therefore, σDD can be reflected by the distribution range of soft faults, and hence it will not be specifically investigated hereafter. In terms of the hard faults, the SA0 and SA1 faults are assumed to be equal in number and they are randomly distributed among all the memristors. The hard fault ratio (γHF) is defined as the percentage of memristors with SA0 and SA1 faults.

The conductance difference (Gij fi − Gij fi+) is mapped to the weight in the range of [−wmax, +wmax], with wmax set to be 0.2 in this study. Note that any finite values can be used for wmax, but the wmax value should be fixed throughout the simulation unless the Gmax and Gmin values of memristors change. When performing the multifault detection and targeted weight pruning, the threshold values ε and ε′ are varied in different cases. The percentage of pruned weights is denoted by γpw. However, the threshold value εgrad for the weight update is fixed to be 0.002.

To make a comprehensive comparison, several controls are used. The first control is the conventional ex situ training (EXT). The second and third controls are two representative hard fault-tolerant training schemes. One is named the HFTT-1 scheme that combines the heuristic algorithm-based weight mapping with targeted weight pruning[18] while the other is named the HFTT-2 scheme that combines online fault detection with fault-blind weight pruning[20] Both HFTT-1 and HFTT-2 schemes use the ex situ training. In addition, in situ training-only schemes using the conventional gradient descent learning rule (ISTGD) and the Manhattan update rule (ISTGD) are used as another two controls. The last control combines ISTGD with multifault detection and targeted weight pruning, which is called the ISTGD+ scheme. Note that combining ISTM with multifault detection and targeted weight pruning indeed forms the proposed MFTT scheme. In addition, the difference between MFTT and ISTGD+ is that the former uses the Manhattan update rule, while the latter uses the conventional gradient descent learning rule.

All the schemes use a minibatch training with a batch size of 250 and 30 training epochs. For EXT, HFTT-1, and HFTT-2, the optimal weights obtained during ex situ training are mapped onto the memristors to generate the highest test accuracy. In contrast, for ISTGD, ISTGD+, ISTM, and MFTT, the test accuracies are averaged over different epochs and random fault distributions. Note that unless otherwise specified, the accuracy mentioned in the rest of this article refers to the classification accuracy on the test set of MNIST.

4.2. Results and Analysis

The MFTT scheme and the control schemes are first implemented on the SLP for the cases where only one type of fault exists (Figure 5, 6, 7, 8 and 9). Figure 5 shows the accuracies of various schemes achieved in the case where only hard faults exist. As assumed earlier, the hard faults contain an equal amount of SA0 and SA1 faults and are randomly distributed among all memristors. Let’s first look at the two schemes designed specifically for the hard faults: HFTT-1 and HFTT-2. HFTT-1 achieves higher accuracies than HFTT-2 in the whole investigated range of γHF, consistent with that observed in the study by Jin et al.[18] Why HFTT-1 outperforms HFTT-2 is because HFTT-1 uses the targeted weight pruning, while
HFTT-2 uses the fault-blind weight pruning, and the former pruning method is more effective to tolerate hard faults.

Then, let us turn to the in situ training algorithms (ISTGD, ISTGD+, ISTM, and MFTT). Although they are not designed specifically for the hard faults, and they still possess some hard-fault tolerance because of the self-adaptivity of in situ training. Moreover, MFTT and ISTGD+ achieve higher accuracies than ISTM and ISTGD, respectively, indicating that the targeted weight pruning can help to improve the hard-fault tolerance. This is not unexpected because all the weights mapped onto the memristors with hard faults are pruned (except C8–C9 shown in Table 1) and no attempts will be made to update these weights; therefore, errors caused by these weights can be minimized. Adjusting only the weights surrounding the pruned ones is sufficient to achieve high accuracy. Figure 5 also shows that MFTT exhibits higher accuracies than ISTGD+, revealing the advantage of the Manhattan update rule over the conventional

![Figure 5](image.png)

**Figure 5.** Accuracies of different schemes as a function of $Y_{HF}$.

![Figure 6](image.png)

**Figure 6.** a) Accuracies of MFTT and ISTGD+ versus $Y_{pw}$ at a given $v_{max}$ of 10. b) Accuracies of various schemes as a function of $v_{max}$. $[0, v_{max}]$ represents the range where the $v$ values of the memristors are distributed.

![Figure 7](image.png)

**Figure 7.** a) Accuracies of MFTT and ISTGD+ versus $Y_{pw}$ at a given $n_{s_{min}}$ of 2. b) Accuracies of various schemes as a function of $n_{s_{min}}$. $[n_{s_{min}}, 32]$ represents the range where the $n_{s}$ values of the memristors are distributed.
gradient descent learning rule. The Manhattan update rule uses the minimum step size to update the adjustable weights to better accommodate the pruned and stuck weights. However, the conventional gradient descent learning rule may introduce excess weight movements, thus lowering the accuracy.

We next focus on the comparison between MFTT and HFTT-1. At $\gamma_{HF} \leq 10\%$, HFTT-1 exhibits higher accuracies than MFTT. In contrast, as $\gamma_{HF}$ exceeds 10%, the accuracies of HFTT-1 are surpassed by those of MFTT. The accuracy losses of HFTT-1 and MFTT at $\gamma_{HF} = 20\%$ are specifically calculated and compared, using the accuracy obtained from fault-free memristors (i.e., 91.6%) as the reference. The accuracy loss of HFTT-1 at $\gamma_{HF} = 20\%$ is 3.9%, which is close to that reported previously. When MFTT is used, an even lower accuracy loss, that is, 2.7%, is obtained. The reason why MFTT can achieve a lower accuracy loss than HFTT-1 at a large $\gamma_{HF}$ is better than that of HFTT-1.

While it has been demonstrated above that MFTT can well tolerate hard faults, how it performs in the presence of soft faults is of great interest. The writing nonlinearity ($\nu$) is the first types of soft faults to be investigated. It is assumed that all the memristors have a uniform distribution of $\nu$ values in the range of $[0, \nu_{max}]$. $\nu$ is defined following a memristor behavioral model suggested by Long-term potentiation (LTP): $G_{actual} = B (1 - e^{-\nu_{p}^{2}}) + G_{min}$ (9)

Long-term depression (LTD): $G_{actual} = G_{max} - B \left(1 - e^{-\nu^{2}}\right)$ (10)

where $G_{actual}$ is the conductance at the pulse number $P$, $G_{min}$ is the minimum conductance, $G_{max}$ is the maximum conductance,
\( v \) is the writing nonlinearity, \( B \) is a conductance range-related parameter equaling \((G_{\text{max}} - G_{\text{min}}) / (1 - e^{-v})\), and \( P_i \) is the total number of pulses used to tune the conductance from \( G_{\text{min}} \) to \( G_{\text{max}} \) (or from \( G_{\text{max}} \) to \( G_{\text{min}} \)). As seen from Equations (9) and (10), the larger \( v \), the more nonlinear the LTP and LTD curves will be, and \( v = 0 \) indicates the linear case.

Using the fault detection with a threshold value \( \epsilon \), memristors with \( v > v_{\text{th}} \) can be detected (as described in Section 3.2). Then, the weights mapped onto memristors with \( v > v_{\text{th}} \) are pruned by setting the conductances of the memristor pairs to \( G_{\text{max}} \) (as described in Section 3.3). It is noteworthy that if either one memristor in the pair has \( v > v_{\text{th}} \), the corresponding weight is pruned. By decreasing \( \epsilon \), \( v_{\text{th}} \) is reduced and consequently the ratio of pruned weights (\( \gamma_{\text{pw}} \)) increases (see Figure S1, Supporting Information, for details).

Figure 6a illustrates the accuracy evolution with varying \( \gamma_{\text{pw}} \) for MFTT and ISTGD+ at a given \( v_{\text{max}} \) of 10 for the distribution range of \([0, v_{\text{max}}]\). For both schemes, the accuracy first increases and then decays as \( \gamma_{\text{pw}} \) increases, forming a peak at a certain \( \gamma_{\text{pw}} \). Why the accuracy first rises to a peak and then falls with increasing \( \gamma_{\text{pw}} \) (i.e., decreasing \( \epsilon \)) can be explained as follows. With the decrease in \( \epsilon \), \( v_{\text{th}} \) is gradually reduced; consequently, weights with large \( v \) are first pruned, followed by those with small \( v \). Weights with large \( v \) are detrimental to the accuracy of the NN because they are typically far away from the target values after programming and thus cause large errors. Pruning them can therefore lead to the increase in accuracy by taking use of the sparsity of NN. In contrast, weights with small \( v \) can function almost normally. Further pruning these weights causes the decrease in accuracy because there are no sufficient number of weights remaining to maintain the proper operation of the NN. Therefore, pruning an appropriate number of weights with large \( v \) is beneficial to the accuracy, but pruning too many weights will deteriorate the accuracy.

Similar accuracy evolutions of MFTT and ISTGD+ with \( \gamma_{\text{pw}} \) are observed for other distribution ranges of \([0, v_{\text{max}}]\), and the optimal \( \gamma_{\text{pw}} \) values leading to peak accuracies are shown in Figure S2, Supporting Information. The peak accuracies of MFTT and ISTGD+ are plotted against \( v_{\text{max}} \) in Figure 6b, along with the accuracies of ISTM, ISTGD, and EXT (note: because HFTT-1 and HFTT-2 can address only hard faults, their performances on soft faults are indeed the same as that of EXT and thus not shown hereafter). At a small \( v_{\text{max}} \) (e.g., \( v_{\text{max}} = 2 \)), MFTT, ISTGD+ + , ISTM, and ISTGD achieve almost the same accuracy (around 90%), suggesting that small \( v \) can be well tolerated by the in situ training. However, EXT achieves a lower accuracy (88.8%), probably because even small \( v \) can cause sizable errors at the weight loading step of ex situ training. As \( v_{\text{max}} \) increases from 2 to 10, the accuracies of ISTM and ISTGD degrade significantly, while those of MFTT and ISTGD+ decrease smoothly and remain at \( > 85\% \). It is therefore demonstrated that the pruning can effectively mitigate the accuracy loss caused by large \( v \) Figure 6b further shows that MFTT achieves a higher accuracy than ISTGD+ at a large \( v \). This is probably because the \( v \)-induced deviation in weight update can be minimized by the Manhattan update rule. Therefore, by combining both pruning and Manhattan update rule, MFTT achieves the best tolerance against \( v \) among all the investigated schemes.

Similar to \( v \), other three types of soft faults, that is, limited number of conductance of states \( (n_s) \), limited dynamic range \((G_{\text{max}}/G_{\text{min}})\), and writing noise \( \sigma_{\text{CIC}} \), are investigated separately. The \( n_s \), \( G_{\text{max}}/G_{\text{min}} \), and \( \sigma_{\text{CIC}} \) values are all assumed to have uniform distributions, and the detailed experimental settings are presented in Supplementary Note 1, Supporting Information. Figure 7a, 7b, and 9a show the accuracy evolutions with respect to \( \gamma_{\text{pw}} \) for MFTT and ISTGD+ at a given \( n_s \) of 2, \((G_{\text{max}}/G_{\text{min}})\) of 1.37, and \( \sigma_{\text{CIC}} \) of 0.5, respectively. Accuracy peaks at certain \( \gamma_{\text{pw}} \) are observed for all the three types of soft faults, similar to that observed for \( v \) (see Figure 6a). The formation origins for these accuracy peaks can also be explained in a similar way. In brief, pruning weights with large soft faults (small \( n_s \), small \( G_{\text{max}}/G_{\text{min}} \), and large \( \sigma_{\text{CIC}} \)) first is beneficial to accuracy, but further pruning weights with small soft faults (large \( n_s \), large \( G_{\text{max}}/G_{\text{min}} \), and small \( \sigma_{\text{CIC}} \)) can deteriorate the accuracy.

Figure 7b summarizes the peak accuracies of MFTT, ISTGD+, ISTM, ISTGD, and EXT with \( n_{\text{min}} \) varying from 2 to 20 for the distribution range of \([n_{\text{min}}, 32]\). For all the investigated \( n_{\text{min}} \) values, the accuracies of MFTT and ISTGD+ are above 87.4% and decrease only slightly with decreasing \( n_{\text{min}} \). In contrast, both ISTM and ISTGD show a dramatic accuracy decay when \( n_{\text{min}} \) becomes smaller than certain values (6 and 9 for ISTM and ISTGD, respectively). These results demonstrate that pruning is particularly useful to suppress the accuracy loss caused by small \( n_s \). In addition, MFTT achieves a higher accuracy than ISTGD+, which can be attributed to the use of the Manhattan update rule. For the Manhattan update rule used in MFTT, the minimum magnitude of weight update is always used; however, for the conventional gradient descent learning rule used in ISTGD+, the magnitude of weight update may be exaggerated for small \( n_s \), thus causing accuracy loss.

Figure 8b shows the peak accuracies of various schemes with varied \((G_{\text{max}}/G_{\text{min}})\) for different distribution ranges of \([G_{\text{max}}/G_{\text{min}}]\). As clearly seen, MFTT and ISTGD+ exhibit higher accuracies than ISTM and ISTGD, respectively, indicating that pruning can reduce the accuracy loss induced by small \((G_{\text{max}}/G_{\text{min}})\). In addition, the accuracy of MFTT is higher than that of ISTGD+, which may be ascribed to the merit of the Manhattan update rule.

Figure 9b shows the peak accuracies of various schemes tested with varying \( \sigma_{\text{CIC}} \) for the distribution ranges of \([0, \sigma_{\text{CIC}}]\). At all the investigated \( \sigma_{\text{CIC}} \) values, MFTT and ISTGD+ exhibit higher accuracies than ISTM and ISTGD, respectively, confirming the effective role of pruning in mitigating the accuracy loss caused by large \( \sigma_{\text{CIC}} \). However, accuracies of MFTT and ISTGD+ are not much different, probably because noises can cause random deviation in the descent direction which can hardly be addressed by both the Manhattan update rule and the conventional gradient descent learning rule.

Having demonstrated the good performance of the MFTT scheme in tolerating only hard faults or one type of soft faults, how it performs when hard faults and various types of soft faults are present simultaneously is of great interest. For the case where both hard and soft faults coexist, the performances of MFTT and...
other schemes are evaluated on both SLP and MLP. The hard and soft faults are assumed to be distributed randomly in the all memristors. The hard fault ratio $γ_{HF}$ is assumed to be 10%, and the distribution ranges of $ν$, $n_e$, $G_{max}/G_{min}$, and $σ_{CIC}$ are assumed to be [0, 10], [18, 32], [2.7, 10], and [0, 0.1], respectively. As shown in Figure 10, HFTT-1, HFTT-2, and EXT exhibit rather low accuracies on both SLP and MLP because they cannot handle the soft faults. The accuracies of ISTG and ISTG on both SLP and MLP are also unsatisfactory, probably due to their limited tolerance against hard and large soft faults. Notably, MFTT and ISTG+ rank top two among all the investigated schemes on both SLP and MLP, demonstrating that the combined multi-fault detection, targeted weight pruning, and in situ training can well address both hard and large soft faults using the sparsity and self-adaptivity of NN. Moreover, MFTT achieves higher accuracies than ISTG+ on both SLP and MLP, suggesting the good tolerance of the Manhattan update rule against the remaining small soft faults. It is therefore demonstrated that the MFTT scheme can tolerate both hard and soft faults simultaneously, which is further verified with different experimental settings (see Figure S3, Supporting Information). This capability distinguishes MFTT from most previous fault-tolerant schemes which could address only hard or soft faults. However, the effectiveness of MFTT on more complex NNs is a question, which will be investigated in further research.

Last but not the least, the multifault detection and targeted weight pruning methods used in MFTT are simple, which would not introduce large hardware overhead. The Manhattan update rule used in MFTT also simplifies the hardware implementation as it requires no computational resources to calculate the applied pulse numbers. In addition, there are no needs to update the pruned weights and the weights with small $|∂J/∂w_j|$ during the weight update step, which can significantly reduce the energy consumption. For example, in the case shown in Figure 10, an average of 6690 weights (total number of weights: 7840) is not updated during the 30 training epochs in MFTT, thereby reducing the writing energy by 85.3%. Therefore, the MFTT scheme provides a hardware-friendly and low-power solution to mitigate the accuracy losses caused by both hard and soft faults for non-ideal memristive NNs.

**5. Conclusion**

In this article, we have proposed an MFTT scheme for addressing both hard and soft faults simultaneously in memristor crossbar-based NNs. The proposed scheme consists of three steps: multifault detection, targeted weight pruning, and in situ training with the Manhattan update rule. The pruning of the detected faulty weights with hard and large soft faults followed by in situ training can effectively tolerate these faults, by making use of the sparsity and self-adaptivity of NN. In addition, the remaining small soft faults can be well tolerated by the Manhattan update rule. Experiments show that in the hard fault-only case, MFTT achieves smaller accuracy losses than previous hard fault-tolerant schemes when the hard fault ratio is large (>10%). In the soft fault-only case, MFTT outperforms previous in situ training-only schemes, particularly when the soft faults are large. When both hard and soft faults are present simultaneously, MFTT achieves the highest accuracy among all investigated schemes. Therefore, the proposed MFTT scheme has good tolerance against both hard and soft faults, making it promising to be applied in the memristor crossbar-based NNs.

**Supporting Information**

Supporting Information is available from the Wiley Online Library or from the author.

**Acknowledgements**

This work was supported by the National Key Research Program of China (nos. 2016YFA0201002 and 2016YFA0300101), Science and Technology Program of Guangzhou (no. 2019050001), National Natural Science Foundation of China (nos. U1931215 and 11674108), and Natural Science Foundation of Guangdong Province (no. 2020A1515010996).

**Conflict of Interest**

The authors declare no conflict of interest.

**Data Availability Statement**

The data that support the findings of this study are available from the corresponding author upon reasonable request.

**Keywords**

fault tolerance, Manhattan update rule, memristive neural network, multifault detection, targeted weight pruning

Received: November 17, 2021
Revised: February 2, 2022
Published online: February 24, 2022

[1] C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, in Thirty-first AAAI conference on artificial intelligence, AAAI Press, San Francisco, 2017, pp. 4278–4284.

[2] K. Simonyan, A. Zisserman, arXiv preprint arXiv:1409.1556, 2014.
[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, IEEE Trans. Pattern Anal. 2017, 40, 834.

[4] J. Redmon, A. Farhadi, arXiv preprint arXiv:1804.02767, 2018.

[5] J. D. Kendall, S. Kumar, Appl. Phys. Rev. 2020, 7, 011305.

[6] T. Gokmen, Y. Visov, Front Neurosci. 2016, 10, 333.

[7] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, H. Qian, Nature 2020, 577, 641.

[8] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, Nat. Mach. Intell. 2019, 1, 434.

[9] L. Song, X. Qian, H. Li, Y. Chen, in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, Piscataway, NJ 2017, pp. 541–552.

[10] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, Y. Xie, ACM Comm. Ar. 2016, 44, 27.

[11] M. Zhao, B. Gao, J. Tang, H. Qian, H. Wu, Appl. Phys. Rev. 2020, 7, 011301.

[12] P.-Y. Chen, B. Lin, I.-T. Wang, T.-H. Hou, J. Ye, S. Vrudhula, J. S. Seo, Y. Cao, S. Yu, in IEEE/ACM Conference on Computer-Aided Design (ICCAD), IEEE, Piscataway, NJ 2015, pp. 194–199.

[13] S. Agarwal, S. J. Plimpton, D. R. Hughart, A. H. Hsia, I. Richter, J. A. Cox, C. D. James, M. J. Marinella, in 2016 International Joint Conference on Neural Networks (IJCNN), IEEE, Piscataway, NJ 2016, pp. 929–938.

[14] Z. Wen, D. Wu, Adv. Mater. 2020, 32, 1904123.

[15] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, L. Jiang, in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, Piscataway, NJ 2017, pp. 19–24.

[16] C. Liu, M. Hu, J. P. Strachan, H. Li, in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, Piscataway, NJ 2017, pp. 1–6.

[17] R. Rajasuman, IEEE Des. Test. Comput. 2001, 18, 16.

[18] S. Jin, S. Pei, Y. Wang, in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, Piscataway, NJ 2020 pp. 91–96.

[19] Z. Song, Y. Sun, L. Chen, T. Li, N. Jing, X. Liang, L. Jiang, IEEE Trans. Comput. Aid. D 2020, 40, 129.

[20] L. Xia, M. Liu, X. Ning, K. Chakrabarty, Y. Wang, in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, Piscataway, NJ 2017, pp. 1–6.

[21] H. Lee, K. K. Likharev, in the 2006 IEEE International Joint Conference on Neural Network Proceedings, IEEE, Piscataway, NJ 2006, pp. 2749–2756.