A Fault-Tolerant Neural Network Architecture

Tao Liu
Florida International University
tliu023@fiu.edu

Wujie Wen
Florida International University
wwen@fiu.edu

Lei Jiang
Indiana University Bloomington
jiang60@iu.edu

Yanzhi Wang
Northeastern University
yanz.wang@northeastern.edu

Chengmo Yang
University of Delaware
chengmo@udel.edu

Gang Quan
Florida International University
gang.quan@fiu.edu

ABSTRACT

New DNN accelerators based on emerging technologies, such as resistive random access memory (ReRAM), are gaining increasing research attention given their potential of "in-situ" data processing. Unfortunately, device-level physical limitations that are unique to these technologies may cause weight disturbance in memory and thus compromising the performance and stability of DNN accelerators. In this work, we propose a novel fault-tolerant neural network architecture to mitigate the weight disturbance problem without involving expensive retraining. Specifically, we propose a novel collaborative logistic classifier to enhance the DNN stability by redesigning the binary classifiers augmented from both traditional error correction output code (ECOC) and modern DNN training algorithm. We also develop an optimized variable-length "decode-free" scheme to further boost the accuracy under fewer number of classifiers. Experimental results on cutting-edge DNN models and complex datasets show that the proposed fault-tolerant neural network architecture can effectively rectify the accuracy degradation against weight disturbance for DNN accelerators with low cost, thus allowing for its deployment in a variety of mainstream DNNs.

ACM Reference Format:
Tao Liu, Wujie Wen, Lei Jiang, Yanzhi Wang, Chengmo Yang, and Gang Quan. 2019. A Fault-Tolerant Neural Network Architecture. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3316781.3316819

1 INTRODUCTION

Deep learning has nowadays achieved phenomenal successes in many real-world applications spanning from computer vision, speech recognition, object detection to game playing and self-driving vehicles [19, 22]. To facilitate DNN's adoption in resource-constrained devices and tackle the significant computation and data movement overhead, many research efforts have been put on developing high-performance and energy-efficient DNN accelerators, such as domain-specific FPGAs, CMOS, and non-CMOS based ASICs [6, 18]. Among the non-CMOS based accelerators, one promising solution is the emerging resistive random access memory (ReRAM or memristor) which integrates both computation and storage simultaneously within the same crossbar array. The key element of DNN computation–multiply-accumulate (MAC) operation can be efficiently conducted within the memristor array by exploiting the relationship between a dot product computation and the currents in a resistive mesh [1]. Many memristor-based DNN accelerators have been proposed [18, 21], and extensive optimizations have been carried out [15, 16]. With the highly paralleled computing architecture and zero cost in data movement, these designs significantly improve the performance-per-watt of DNN accelerators, far exceeding that of CMOS-based counterparts.

However, one critical challenge faced by these memristor-based accelerators is their poor stability. A DNN weight, which is represented as the memristance of a memristor cell, can be easily distorted by the inherent physical limitations of memristor devices [4, 17]. For example, the electrical or thermal noise and process variations can limit the programming precision of a memristor. The endurance varies widely from cell to cell and from device to device, causing highly imbalanced wear-leveling. Memristance drift [4] induces tiny perturbations on memristance states which in turn degrade the DNN computing accuracy, performance, and system stability [23]. Although recent works have investigated errors in ReRAM accelerators [5, 14], their solutions focus on permanent defects (i.e., stuck zero or one fault), overlooking the far more common noise, drifting, and programming errors these devices are likely to encounter. More importantly, their solutions for tolerating defects usually involve non-trivial retraining, which is far from scalable in the envisioned scenario of a neural network trained once in the cloud and deployed to many edge devices each equipped with a ReRAM accelerator. Furthermore, each ReRAM accelerator displays a unique footprint of defects and errors due to process variations and aging, multiplying their proposed efforts dramatically.

Fundamentally different from these approaches, our solution proposed in this paper intends to address the stability problem *without involving expensive retraining, but exploiting and further boosting DNN's self-correcting capability.* The inherent error resilience of DNNs, which already allows it to handle minor precision loss and data errors [10, 13], can be escalated by wisely redesigning the ensemble learning method such as error-correcting output code (ECOC) [8] for modern DNNs. However, boosting the error-resilience capability of DNNs to the level capable of mitigating weight disturbance in ReRAM accelerators needs to resolve two major technical challenges: 1) Modern DNN classifier usually uses softmax regression in the output layer to solve the mutually exclusive multi-class classification problem (winner-takes-all rule by

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
DOI: 10.1145/3316781.3317742

one-hot encoding), which is incompatible with ECOC. Directly replacing the softmax classifier with a set of independent binary logistic classifiers will cause undesired accuracy loss because of the increased neural competition [16]; 2) Existing ECOC [3] requires comprehensive training on the classifier dedicated to a certain task, making such a solution inflexible when handling a variety of machine learning tasks.

To overcome these challenges, in this work, we investigate and propose a set of techniques to unleash the algorithmic error-resilience of DNN classifier. A critical observation is that while small weight disturbances may occur in any layer of a given DNN model, propagate through the network, and introduce variations to the score of each class on the output layer, they affect the final classification outcome **if and only if** the ranking of different classes on the output layer is altered. Based on this observation, our work targets the output layer and enhances DNN stability with a collaborative logistic classifier which leverages asymmetric binary classification coupled with an optimized variable-length decode-free ECOC to improve the error-correction capability of DNN accelerators. Our scalable design requires neither expensive defect-map-specific calibration nor training-from-scratch, and can be easily integrated with existing hardware-based fault tolerance solutions. Extensive experimental studies on different DNN models and datasets confirm that our design significantly reduces the neural competition and increases the decision (Hamming) distance on final classification output, thus effectively rectifying the accuracy degradation induced by resistance variations and stuck-at faults (SAFs) in emerging ReRAM accelerators.

2 BASICS OF DNN AND ECOC

Deep neural network (DNN) is usually composed of different types of layers. The convolutional layer abstracts features from the inputs through the kernel-based convolutions. The fully-connected layer further ranks the confidence of each class based on the weighted features. The output layer is used as the DNN classifier such as softmax and logistic to make the final decision.

2.1 Logistic and Softmax Classifier

The logistic classifier is a classic solution to solve the traditional binary classification problem (e.g., true or false). Given input features $x^{(i)} \in \mathcal{R}^n$ and neural network weights θ, the logistic classifier can be trained with label $y^{(i)} \in \{0, 1\}$ through logistic regression $h_\theta(x)$ with gradient $\nabla_\theta J(\theta)$:

$$\begin{cases} h_\theta(x) = 1/1+\exp(-\theta^\top x) \\ \nabla_\theta J(\theta) = -\sum_i x^{(i)}(y(i) - h_\theta(x^{(i)})) \end{cases} \tag{1}$$

To handle the complex multi-class classification [7], softmax classifier is widely adopted in modern DNNs.

$$\begin{cases} h_\theta(x) = \exp(\theta^{(k)\top}x))/\sum_{j=1}^{K}\exp(\theta^{(j)\top}x) \\ \nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left(x^{(i)}\left(\{y^{(i)} = k\} - h_\theta(x)\right)\right) \end{cases} \tag{2}$$

Based on the one-hot coding (i.e., label $y^{(i)} = 1$ for target class and $y^{(i)} = 0$ for others), softmax classifier can push the gradient towards the target class by normalizing the multiple output logits, thereby achieving better accuracy than logistic classifier.

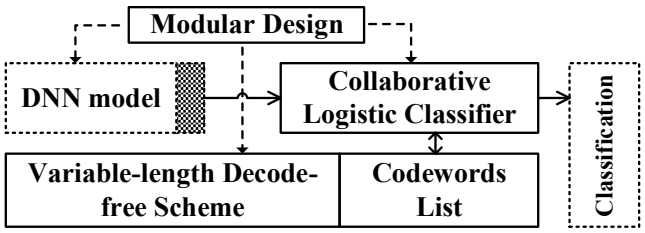

Figure 1: Overview of fault-tolerant neural network architecture.

2.2 Error-Correcting Output Code (ECOC)

As the logistic classifier suffers from limited predictive performance on multi-class classification, ECOC is an ensemble learning method to address this issue [25]. Due to the independence among logistic classifiers, neural network outputs can be treated as a specific codeword. Therefore, ECOC can solve the multi-class classification as the traditional coding problem: given input features $x^{(i)} \in \mathcal{R}^n$, L independent logistic classifiers can be trained with a $K \times L$ coding matrix $M_{(K,L)}$, where codeword $M_{(i,L)}$ is assigned with a L-dimension label vector $Y_L^{(i)} \in \{1, 2, \cdots, K\}$. Particularly, the l-th logistic classifier can be trained with label $y_l^{(i)} \in \{0, 1\}$ by following Eq. 1. Based on this learning scheme, appropriate error-correcting coding (e.g., Hamming) or optimized coding matrix can be further applied to ECOC to increase Hamming distance of the codewords assigned for different classes (i.e., enlarge the margin of decision boundary and reduce the complexity of classification) [8], thus to enhance the accuracy.

3 OUR DESIGN

3.1 Overview

Fig. 1 depicts an overview of the proposed fault-tolerant neural network architecture (FTNNA). In FTNNA, the collaborative logistic classifiers mainly focus on improving the classification results based on the significance of each logistic classifier, while the coding scheme further addresses the neural competition issue among different classes by using the variable-length asymmetric coding/decoding manner. The coding scheme can be established through the proposed searching code, to generate a codeword list used for classification.

Modular design. FTNNA can be implemented through a modular design with improved scalability. Given any DNN model, 1) We first test the original accuracy and collect the DNN confusion matrix; 2) The original classifiers (i.e., softmax) in the output layer of the given model will be replaced by a certain number of collaborative logistic classifiers, which is fully connected to the previous layer; 3) The confusion matrix and number of collaborative logistic classifiers will be sent to the DNN-favorable searching code to create the codeword list; 4) The weights of collaborative logistic classifiers will be fine-tuned through transfer learning [24] on a given dataset based on the codeword list.

3.2 Collaborative Logistic Classifier

In FTNNA, the proposed collaborative logistic classifier is extended from the traditional logistic classifier to handle the ECOC based classification. To solve the aggravated neural competition issue and

improve the error-correction capacity, our basic idea is to introduce some dependencies among different classifiers (as softmax classifier) in fine-tuning, while maintaining the independence of binary classifier for ECOC coding. To achieve such a goal, we leverage the fine-tuning and regression algorithms.

Fine-tuning. Fine-tuning technique such as transfer learning [24] is usually applied on pre-trained DNN models to handle different classification tasks. Such a technique will only update the weights of DNN classifiers thus to rebuild the decision making without expensive training. Inspired by the softmax classifier, we design the collaborative logistic classifier with improved dependency by leveraging the gradient descent based fine-tuning algorithm.

To fine-tune the weights of the collaborative logistic classifier, we still use the logistic regression as presented in Eq. 1. However, to increase the dependency among the classifiers, we introduce the significance parameter set $\{\beta\}$ and assign the significance on each classifier to establish the correlations among logistic classifiers:

$$\nabla_\theta J(\theta) \propto -\beta_{(k)} \cdot \sum_i x^{(i)}(y_{(k)}(i) - h_\theta(x^{(i)})) \tag{3}$$

In our implementation, to simplify the approach and better control the pace of weight update, a regularization term $\sigma(x, \theta)$ is applied on the loss function $\mathcal{L}$ to rectify the classifier significance during fine-tuning:

$$\nabla_\theta \left(1/n \sum \mathcal{L}(y, h_\theta(x)) + \sigma(x, \theta) \right) \tag{4}$$

Specifically, the regularization term $\sigma(x, \theta)$ is calculated based on the Hamming distance of the corresponding classifier's target codeword and its predicted one. since the neural competition can be estimated from the bit-flipping occurrences, after the fine-tuning, more significant classifiers may give more decisive confidence for decision making.

Regression. Due to the rectified significance, some classifiers may again become indecisive during inference. Therefore, We further set a pending zone in logistic regression to address this issue. The pending zone is defined as a specific region:

$$h_\theta(x) = 1/1+\exp(-\theta^\top x)|_{-0.4 \le \theta^\top x \le 0.4} \approx [0.4, 0.6] \tag{5}$$

Once the weighted input $\theta^\top x$ enters the pending zone, the classifier will report both $\{0, 1\}$ as its output. For example, given three collaborative logistic classifiers with an input vector $\{-2, 0.1, 2\}$, the output vector(s) will be a 2-dimension matrix $\{0, 0, 1\}$ and $\{0, 1, 1\}$. Later, the Hamming distance of these two codewords will be compared with the entries in codeword list to predict the target class. Such a design may effectively rectify the wrong decisions caused by less significant classifiers. However, there also exists a rare case that multiple codewords can output the same Hamming distance after comparisons. To handle this issue, a simple solution is to temporally disable the pending zone, i.e., using its original threshold (i.e., $\theta^\top x = 0.5$) for regression, once the most significant classifier with current input enters the pending zone.

3.3 Coding Scheme

To design the low-cost, DNN-favorable ECOC, we propose the variable-length decode-free coding scheme with searching code to further reduce the neural competition by leveraging the asymmetric decoding in FTNNA. To minimize the decoding cost, our coding scheme will create the codeword list. Such a design may require some efforts during encoding phase, but can significantly reduce the decoding cost by only checking the predefined lookup table (LUT). Once the DNN-favorable searching code finishes the encoding, the output codeword list will be stored in the LUT and will be later accessed by collaborative logistic classifiers for classification, which can be performed by comparing the hamming distance (through XOR). This leads to a low-cost decode-free design.

Variable-length coding. The variable-length coding attempts to further alleviate the neural competition to improve the accuracy. For ECOC coding, a simple solution is to aggressively increase the number of classifiers (i.e., enlarge the coding bit-length), which may even surpass that of original softmax classifiers with one-hot coding. However, this design may significantly increase the overhead of DNN accelerator and is also unnecessary as the accuracy can be saturated once the bit-length reaches a certain value (see Sec. 4). To achieve the variable-length coding, we follow a similar constraint from the general searching code [12] and take the DNN confusion matrix into consideration: 1) The codeword should separate from each other with the maximized hamming distance; 2) The bit-column (i.e. the binary combination of a certain classifier on the same bit position) should have the maximum hamming distance from each other; 3) The codeword should separate the most overlapped classes in confusion matrix; 4) There are no all-1, all-0 and complementary bit-columns.

DNN-favorable searching code. The pseudo-code of proposed DNN-favorable searching code is described in Algorithm 1, which mainly consists of three parts: prepare searching table (line 1-8), searching code (line 9-15) and code assign (line 16-24). The prepared searching table indicates the maximum number of possible

Algorithm 1: DNN-favorable searching code

```
1  l ← code length (number of classifiers);
2  h ← Hamming distance;
   // initialize searching table T ← {0}
3  while (T = {0})&&(h ≥ 3) do
4      foreach i ∈ [1, 2^n − 1] do
5          foreach t ∈ T do
               // assess the Hamming distance
6              if Hamm(Dec2Bin(i, l), Dec2Bin(t, l)) ≥ h then
7                  T ← T ∪ {i}
8      if T = {0} then
9          h← h-1
   // searching code
   // initialize output code list O ← {0}
10 m ← number of classes, x ← 1;
11 n = CodeLen(T,h,m);
12 while Sizeof(O) < m&&x < 2^n do
13     foreach o ∈ O do
14         if Hamm(Dec2Bin(o, l), Dec2Bin(x, l)) ≥ h then
15             O ← O ∪ {i}
16     x←x+1;
   // assign searching code to class
17 C ← confusion matrix;
18 j ← m;
19 S ← {}// record assignment
20 while (Sizeof(C) ≥ 0)&&(j ≥ 0) do
       // Pop the max value from confusion matrix
21     c ← Pop(Max(C));
22     if (xindex(c) ≠ yindex(c))&&(xindex(c) ∉ S) then
           // x,y index of confusion element is not equal indicates
           // the current strongest classification error
23         Class(xindex(c))← Dec2Bin(Pop(O));
24         S ← xindex(c) ∪ S;
25         j ← j + 1;
```

Table 1: Experimental Settings.

Environment		
CPU		Intel Core i7-6850K, 12 cores
GPU		GeForce GTX 1080, 2560 CUDA cores
Simulator		MATLAB, Deep Learning Toolbox
Network Model	**Dataset**	**Original Accuracy**
MLP [20]	Mnist	99.1%
LeNet [20]	Cifar-10	76.1%
Alexnet [10]	ImageNet	57.2%
Squeeze [11]	ImageNet	57.5%

codewords with code length l and Hamming distance h, which can be automatically adjusted. The searching code returns a set of codewords that satisfies the aforementioned constraints. In code assign phase, we evaluate the confusion matrix and gradually pick up the corresponding class with strongest neural competition (i.e., with current strongest classification error), which will be assigned with the searched code with highest priority. Note the coding process is done off-line before the neural network accelerators download the well-trained DNN models.

4 EVALUATION

4.1 Experimental Setup

Baselines and benchmarks. Table 1 shows the details of our experimental environment, neural network models and datasets. We select four different neural network models, including small-scaled multi-layer perceptron (MLP) and popular convolutional neural networks (CNN) such as LeNet, Alexnet and Squeezenet, along with three datasets ranging from simple Mnist (10-class handwritten digits), Cifar-10 (10-class tiny images) and complex Imagenet (1000-class large images), so as to comprehensively validate the efficiency and scalability of proposed FTNNA. We simulate a memristive accelerator similar to [9], wherein each layer of selected neural network model is mapped to one or more 128×128 arrays and each memristive cell maintains 64 quantization levels (6-bit) to achieve a good balance between throughput and reliability [2].

Error modeling. Two different types of errors [5] are simulated in our evaluation: stochastic programming error (represented as *resistance variation* in this paper) and stuck-at fault (SAF). The resistance variation can be formulated as:

$$w' \leftarrow w \cdot e^{\theta} \text{ s.t. } \theta \sim N(0, \sigma^2) \tag{6}$$

where w' is the neural network parameters with programming errors under memristor resistance variation θ, which follows a log-normal distribution. In our simulation, we vary σ to change the level of resistance variation, so as to tune random programming error. The SAF occurs when a memristor device freezes in a low resistance state (LRS) or high resistance state (HRS), resulting in the stuck-at-one (SA1) fault or stuck-at-zero (SA0) fault. We adopt SA0 (SA1) fault rate as 1.75% (9.04%) based on the published data [5].

Experimental method. We use the classification accuracy as the measurement metric for our proposed FTNNA [5]. The original accuracy (baseline, without considering device errors) of the accelerators implemented with the four selected neural network models under corresponding datasets, are reported in Table 1, serving as the upper bound of our fault-tolerant design. To characterize the lower bound of the accuracy, selected error models (both programming errors and SAFs) are first applied to the weights across all different layers in selected neural network models. We then further apply our proposed fault-tolerant architecture FTNNA to each weight-distorted neural network model and measure its average accuracy. Monte-Carlo simulations, which perform 1000 times of testing for each combination of neural network model and error model, are conducted to calculate the average accuracy. Particularly, we set the number of collaborative logistic classifiers as $l = 7$ ($l = 500$) for the 10-class Mnist and Cifar-10 datasets (1000-class Imagenet dataset) in our evaluation.

4.2 Results and Analysis

Mitigating programming error (or resistance variation). We first explore how our proposed FINNA can response to the stochastic programming error incurred by resistance variations. Fig. 2 compares the classification accuracy of the four neural network models before (**blue line–Error, baseline**) and after (**green line–FTNNA**) applying the proposed FINNA across different levels of resistance variations (σ). Compared to the four baselines which suffer from severer accuracy degradation as resistance variation σ increases, our proposed FTNNA can always provide significant accuracy improvement for all the models handling small or large datasets, clearly demonstrating the scalability of FTNNA. Impressively, the biggest accuracy gap, i.e. ~ 40%, ~ 50%, ~ 30% and ~ 28% for MLP-Mnist, LeNet-Cifar10, Alexnet-Imagenet and Squeezenet-Imagenet, can be well maintained even when approaching the largest variation $\sigma = 1.5$.

As Fig. 2(a) shows, for small MLP-Mnist, FTNNA achieves > 90% accuracy for a smaller variation ($\sigma < 0.6$), while that of the baseline–error is significantly degraded, e.g. from the ideal level (99%) to very unacceptable level (60%). Although the effectiveness of FINNA slightly decreases at larger variations ($0.8 < \sigma \leq 1.5$), it can still offer ~ 50% better accuracy. Unlike the MLP-Mnist, FTNNA performs even better for complicated convolutional neural networks (CNNs) like LeNet and Alexnet when handling larger datasets–Cifar-10 and Imagenet. As Fig. 2(b) shows, the accuracy degradation (76% → 19% for baseline–Error) on LeNet at variation $0 < \sigma < 1$ can be reduced to merely ≤ 6% under the protection of FTNNA, translating into ~ 50% accuracy improvement. The boosted accuracy is even close to the ideal accuracy (> 70% v.s. 76%). We also find a similar trend for more complex Alexnet-Imagenet, i.e. accuracy is improved by ~ 35% while the ideal accuracy is 57%, as Fig. 2(c) shows. This is because FTNNA can greatly unleash its error correction potential for complex neural networks (like CNNs) with sufficient number of parameters and better error-resilience capability. Moreover, we observe that the improvement of Squeezenet, a highly pruned and compressed DNN model that consists only convolutional layers but offers a similar accuracy to Alexnet, is less significant than Alexnet, e.g. from ~ 15% to ~ 40% v.s. from ~ 15% to ~ 65% at $\sigma > 1$. The reason is because the collaborative classifier in FTNNA can better handle the errors on fully connected layer (for decision making) than that of convolutional layer (for feature extraction). Note Alexnet maintains ~ 90% parameters in fully connected layers.

Mitigating combined error–programming error + SAFs. We also evaluate the efficiency of FTNNA against more severer errors by combining both programming errors and SAFs. As expected, for all selected baseline–Error, the classification accuracy can be further significantly degraded (even at $\sigma = 0$) due to the additional

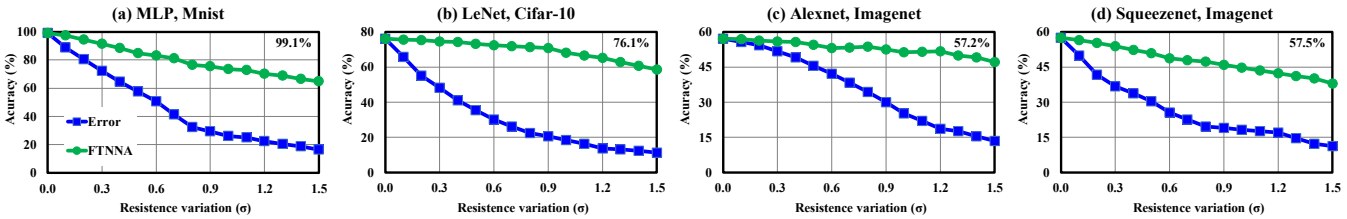

Figure 2: Classification accuracy on selected baselines with resistance variation by varying σ.

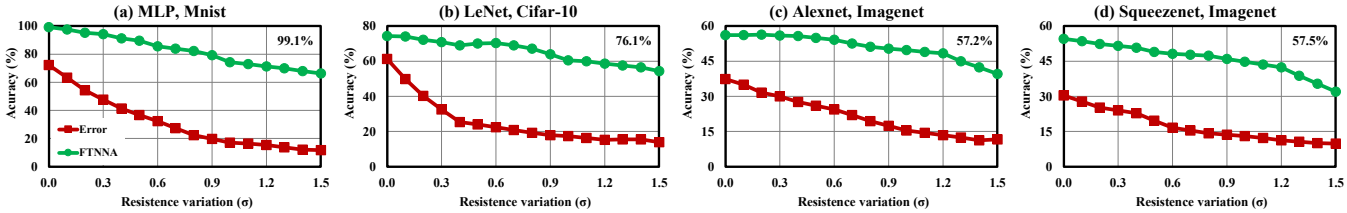

Figure 3: Classification accuracy on selected baselines with resistance variation and SAFs by varying σ.

SAFs (see Fig. 3). However, FTNNA can always effectively restore the accuracy that is close to the ideal level, translating into more significant improvement compared with that of programming error only. For example, FTNNA improves the accuracy from ~ 30% (~ 20%) to ~ 50% (~ 35%) on average for MLP (Alexnet) across the whole range of σ. These results clearly indicate that FTNNA can handle the SAFs more effectively than the programming errors. The reason is because the decision confusion caused by weights with bi-directional SAFs (LRS to HRS or HRS to LRS) can be better alleviated by the variable-length coding/decoding in FTNNA.

4.3 Discussion

Integration with existing solutions. For larger resistance variations ($\sigma > 1.2$), FTNNA still gradually becomes less effective, as Fig. 3 and Fig. 2 show. This is consistent with previous works [5, 9], since the fault-tolerance capability can be eventually compromised by strong variations. However, as an orthogonal solution that well leverages the algorithmic fault-tolerance of neural network classifier, FTNNA can be naturally integrated with existing solutions such as bipartite-matching [5], thus to further improve the robustness. Previous work [5] also shows that combining bipartite-matching and redundancy rows together can better handle programming errors and SAFs on memristive neural network accelerators. Here we integrate our technique into bipartite-matching, to investigate how much redundancy rows FTNNA can save for the same accuracy.

Fig. 4 shows the combined effectiveness of FTNNA and bipartite-matching (named as "ours") on top of bipartite-matching only, against SAFs together with selected resistance variations (i.e., $\sigma = 0.5, 1, 1.5$) on a variety of designs with different number of redundant rows. The MLP-Minst design is selected. For each selected design, FTNNA can always further boost the accuracy, with more significant improvement on designs with fewer number of redundancy rows. For example, FTNNA improves the accuracy by 15%, 16% and 13% with $\sigma = 0.5, 1, 1.5$, respectively, for designs with 20 redundant rows, when compared with bipartite-matching. To show the improvement more clearly, we highlight the best accuracy at each σ offered by bipartite-matching with 100 redundant rows, i.e., 99% with $\sigma = 0.5$ (blue line), 89% with $\sigma = 1$ (red line) and 80% with $\sigma = 1.5$ (green line). With integrated FTNAA, we save more than 50% of redundant rows, i.e., 50, 40 and 50 for $\sigma = 0.5, 1, 1.5$, respectively, in order to achieve the same high accuracy. These results further indicate the improved effectiveness and scalability of proposed FTNNA.

Flexibility. Since the collaborative logistic classifier incorporates variable-length coding scheme, Fig. 5 further evaluates the flexibility of FTNNA by comparing the accuracy of different FTNNA designs with the original ECOC design. The CLC-20 (CLC-100, CLC-500) design consists of 20 (100, 500) collaborative logistic classifiers to classify the 1000 classes in Imagenet dataset, while the ECOC directly uses Hamming code (16 binary classifiers) for classification.

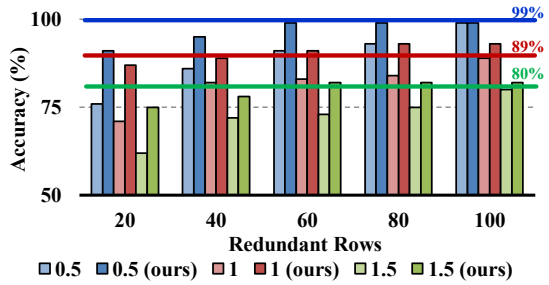

Figure 4: Working with bipartite-matching [5] on MLP-Mnist.

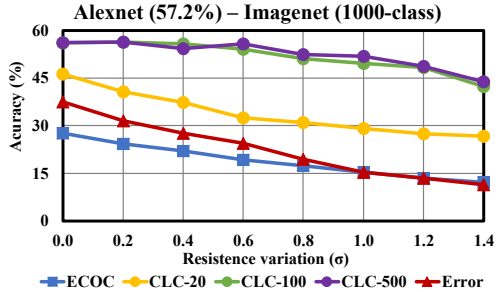

Figure 5: Comparison between ECOC and various CLC designs.

Table 2: The overhead of FTNNA.

		MLP (10-class)	Alexnet (1000-class)
Original	**Parameters**	5.86 KB	3000 KB
FTNNA	**Parameters**	4.10 KB	1500 KB
	LUT	0.0085 KB	61.0352 KB
	Total	4.1085 KB	1561.0352 KB

As shown in Fig. 5, the ECOC is completely ineffective against the SAFs and resistance variation. In fact, the accuracy of ECOC on Alexnet is even worse than the modern softmax classifier with errors, due to the limited classification capability under significantly reduced number of classifiers (i.e., 16 in ECOC v.s. 1000 in softmax). In contrast, our CLC-20 significantly surpasses the ECOC, i.e., increasing the accuracy by 15%, even with only 20 collaborative logistic classifiers. By increasing the number of collaborative logistic classifiers, FTNNA continues improving the classification accuracy (i.e. from CLC-20 to CLC-100) because the increased coding space can alleviate the conflict of coding for different classes. However, the error correction capability of FTNNA can still be saturated when reaching a sufficient number of classifiers, e.g. CLC-500 almost maintains the same level of accuracy as CLC-100.

Overhead. Table 2 compares the storage overhead between the original design of selected neural network models (quantized to 6-bit) and FTNNA. Note that we only evaluate the final layer of selected models, since FTNNA only replaces the original softmax classifiers with collaborative logistic classifiers. FTNNA can actually decrease the storage overhead by 30% (48%) compared with the original design of MLP (Alexnet). The incurred LUT overhead can be very marginal compared with the original design. Due to the reduced number of classifiers (i.e., 7 in MLP and 100 in Alexnet) in FTNNA, the storage requirement of neural network parameters in the last layer can be significantly reduced, which saves the computation overhead especially for the last-layer dominant neural network models. In our simulation, we observe that the classification efficiency (include searching the LUTs) can be improved by $\sim 1.2\times$ ($\sim 1.7\times$) on MLP (Alexnet) with the low-cost collaborative logistic classifiers in FTNNA due to the reduced computation.

5 CONCLUSION

This paper presents a fault-tolerant neural network architecture to tackle the accuracy drop issue of emerging ReRAM based neural network accelerators caused by the resistance variations and stuck-at faults (SAFs) within these devices. The proposed work enhances the algorithm level error-resilience capability of DNN classifiers through a collaborative logistic classifier design by leveraging both asymmetric binary classification and an optimized variable-length "decode-free" scheme. This algorithmic solution is highly cost-effective and scalable, as it does not require expensive defect-map-specific calibration or training-from-scratch. Experimental results show that our design can effectively rectify the accuracy degradation problem on emerging DNN accelerators, and can be easily integrated with existing hardware-based fault tolerance solutions for higher accuracy at lower overhead.

ACKNOWLEDGMENTS

This work is partially supported by NSF Grants CNS-1840813 and CCF-1527464.

REFERENCES

[1] Hiroyuki Akinaga and Hisashi Shima. 2010. Resistive random access memory (ReRAM) based on metal oxides. *Proc. IEEE* 98, 12 (2010), 2237–2251.
[2] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201.
[3] Adam Berger. 1999. Error-correcting output coding for text classification. In *IJCAI-99: Workshop on machine learning for information filtering*.
[4] Ting Chang, Sung-Hyun Jo, and Wei Lu. 2011. Short-term memory to long-term memory transition in a nanoscale memristor. *ACS nano* 5, 9 (2011), 7669–7676.
[5] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. 2017. Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 19–24.
[6] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (2016), 105–112.
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 248–255.
[8] Thomas G Dietterich and Ghulum Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research* 2 (1995), 263–286.
[9] Ben Feinberg, Shibo Wang, and Engin Ipek. 2018. Making memristive neural network accelerators reliable. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 52–65.
[10] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[11] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).
[12] Yan-huang Jiang, Qiang-li Zhao, and Xue-jun Yang. 2004. A general coding method for error-correcting output codes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 648–652.
[13] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.
[14] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. 2017. Rescuing memristor-based neuromorphic design with high defects. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 1–6.
[15] Tao Liu, Lei Jiang, Yier Jin, Gang Quan, and Wujie Wen. 2018. PT-spike: a precise-time-dependent single spike neuromorphic architecture with efficient supervised learning. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 568–573.
[16] Tao Liu, Zihao Liu, Fuhong Lin, Yier Jin, Gang Quan, and Wujie Wen. 2017. MT-spike: a multilayer time-based spiking neuromorphic architecture with temporal error backpropagation. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 450–457.
[17] Gilberto Medeiros-Ribeiro, Frederick Perner, Richard Carter, Hisham Abdalla, Matthew D Pickett, and R Stanley Williams. 2011. Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution. *Nanotechnology* 22, 9 (2011), 095702.
[18] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.
[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[20] Patrice Y Simard, Dave Steinkraus, and John C Platt. 2003. Best practices for convolutional neural networks applied to visual document analysis. In *null*. IEEE, 958.
[21] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 541–552.
[22] Christian Szegedy. 2016. An Overview of Deep Learning. *AITP 2016* (2016).
[23] Bonan Yan, Jianhua Joshua Yang, Qing Wu, Yiran Chen, and Hai Helen Li. 2017. A closed-loop design to enhance weight stability of memristor based neural network chips. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 541–548.
[24] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.
[25] Bin Zhao and Eric P Xing. 2016. Sparse output coding for scalable visual recognition. *International Journal of Computer Vision* 119, 1 (2016), 60–75.