Reliability-Driven Neural Network Training for Memristive Crossbar-Based Neuromorphic Computing Systems

Junpeng Wang[1], Qi Xu[2], Bo Yuan[3], Song Chen[1], Bei Yu[4], Feng Wu[1]
[1]School of Microelectronics, University of Science and Technology of China; USTC Beijing Research Institute, Beijing, China
[2]Department of Electronic Science and Technology, Hefei University of Technology
[3]Department of Computer Science and Engineering, Southern University of Science and Technology
[4]Department of Computer Science and Engineering, The Chinese University of Hong Kong
{wjp97@mail,songch@,fengwu@}ustc.edu.cn, xuqi@hfut.edu.cn, byu@cse.cuhk.edu.hk

***Abstract*—In recent years, memristive crossbar-based neuromorphic computing systems (NCS) have provided a promising solution to the acceleration of neural networks. However, stuck-at faults (SAFs) in the memristor devices significantly degrade the computing accuracy of NCS. Besides, the memristor suffers from the process variations, causing deviation of the actual programming resistance from its target resistance. In this paper, we propose a reliability-driven network training framework for a memristive crossbar-based NCS, with taking account of both SAFs and device variations challenges. A dropout-inspired approach is first developed to alleviate the impact of SAFs. A new weighted error function, including cross-entropy error (CEE), the l_2-norm of weights, and the sum of squares of first-order derivatives of CEE with respect to weights, is further proposed to obtain a smooth error curve, where the effects of variations are suppressed. Experimental results show that the proposed method can boost the computation accuracy of NCS and improve the NCS robustness.**

***Index Terms*—neuromorphic computing system, memristive crossbar, fault tolerance, robustness**

I. Introduction

Neuromorphic computing systems (NCS) based on hardware designs intend to mimic neuro-biological architectures [1]. Different from conventional von Neumann architectures, NCS has been often constructed with highly parallel, extensively connected, and collocated computing and storage units, which eliminate the gap between CPU computing capacity and memory bandwidth [2]. However, the implementation of NCS on CMOS technology suffers from a mismatch between NCS building blocks (neuron and synapse) and CMOS primitives (Boolean logic). Recently, the emerging memristive technology is adopted to implement the synapse circuit thanks to the similarity between memristive and synaptic behaviors [3], [4]. For example, the memristor is suitable to store the weight of synapse because the resistance of a memristor can be programmed by applying current or voltage. Besides, compared with the state-of-the-art CMOS design, the memristive crossbar has been proven as one of the most efficient nanostructures that carry out matrix-vector multiplications while hardware cost and computation energy are significantly reduced [1].

Despite these tremendous advantages, NCS implementations on memristive crossbars encounter some reliability challenges. First, memristors suffer from stuck-at faults (SAFs), which make the memristor stuck at high or low resistance state, leading to a significant yield loss in NCS. Second, in a memristor-based NCS, programming the resistance of the memristors induces stochastic device variations [5]. As a result, the actual programming resistance is deviated from its target resistance and finally results in significant errors to the output of the neural network.

To tolerate SAFs, a number of solutions have been proposed. Huangfu *et al.* [6] proposed a mapping algorithm for tolerating SAFs by using extra memristive crossbars. Xia *et al.* [7] presented a re-mapping scheme with inner fault tolerance to eliminate the impact of

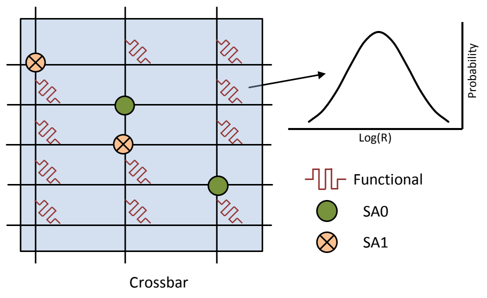


Fig. 1: The memristive crossbar with SAFs and device variations.

SAFs. But redundant memristors are needed in the mapping-based fault tolerant methods, thus inevitably brings hardware overhead. Recently, machine learning techniques have been successfully utilized to address SAFs problem. Liu *et al.* [8] presented a re-training based method to tolerate SAFs. Xia *et al.* [9] proposed a fault tolerant training with re-mapping technique by using the sparsity of neural networks. However, the solutions only focus on SAFs, overlooking the programming device variations.

Besides, to address stochastic device variations, Liu *et al.* [10] developed a variation tolerant training by adjusting the training goal according to the impact of the variations. Thus a set of pre-trained neural networks is generated. By testing the network models, the best one is applied to the memristive crossbar. However, for a crossbar with stochastic variations, the optimal network model can hardly be derived. Chen *et al.* [11] investigated a fault and variation tolerant framework for memristive crossbar-based NCS. It first finds an optimal mapping between weights and memristors for SAF and variation tolerance. Then the re-training process based on conventional gradient descent technique is further adopted to tune weights. But the resistance variation values of memristors are assumed to be known in advance, thus the method is essentially no different from the SAFs-aware approaches.

Although recent works have investigated the SAFs and variations tolerance, the SAFs tolerant solutions can be derived only if the exact locations of SAFs in crossbar are known in advance. Although the SAFs locations in the crossbar can be obtained by March-C algorithm or squeeze-search algorithm based testing methods [12], [13], the test overheads may be too high for the NCS implemented by many memristive crossbars. Therefore, a general reliability design of the memristive crossbar-based NCS without the prior testing is required. Besides, despite a general variation tolerant design of NCS is presented in [10], the impact of SAFs are not considered. Fundamentally different from these approaches, we propose a reliability-driven neural network training framework for a memristive crossbar-based NCS. To the best of our knowledge, this is the first NCS reliability-driven training flow with taking account of both SAFs and variations challenges simultaneously. Key technical contributions of this work are listed as follows.

- We propose a general reliability-driven neural network training algorithm to enhance the robustness of NCS to SAFs and device variations.
- A dropout-inspired approach is developed to alleviate the impact of SAFs.
- A new weighted error function, including cross-entropy error (CEE), the l_2-norm of weights, and the sum of squares of first-order derivatives of CEE with respect to weights, is designed to obtain a smooth error curve, where the effects of device variations are suppressed.
- Experimental results show that our method not only improves the robustness of NCS to SAFs and device variations, but also boosts the computation accuracy of NCS.

The remainder of this paper is organized as follows. Section II presents the preliminary and introduces the problem to be addressed in the paper. Section III describes the proposed reliability-driven design framework for neuromorphic computing systems. Section IV presents experimental results, followed by conclusion in Section V.

II. Preliminaries

A. Fault and Variation Models on Memristor

Due to the immature fabrication technology, manufacturing reliability is still a major concern in a memristive crossbar-based NCS. The memristor faults can be divided into soft faults and hard faults. For a soft fault, the resistance of an RRAM cell is not correct but can still be tuned. However, for hard faults, the resistance of a memristor cannot be changed; thus, the computational accuracy of NCS is limited. Among all kinds of hard faults, stuck-at-one (SA1) faults and stuck-at-zero (SA0) faults appear rather frequently. The permanent open switch defects and broken word-lines lead to SA1 faults which behave that 9.04% of the memristors are in a high resistance state. Meanwhile, the SA0 faults, which are caused by over-forming defects, reset failures, or short defects, force 1.75% of the on-chip memristors in a low resistance state [11], [13].

Various types of variations are observed in memristors, which are divided into two categories, i.e., parametric variation and switching variation. The device-to-device parametric variation is caused by fabrication imperfection, such as random discrete dopants and the line edge roughness [14]. On the other hand, the switching variation is a cycle-to-cycle variation triggered by driving circuit design. Any small fluctuations in the magnitude and pulse-width of the programming current or voltage can produce resistance variations. Throughout this paper, we use matrix $\boldsymbol{C}$ to represent the resistance states of the crossbar. The work in [15] have reported that the programmed resistance states of the memristors follow a lognormal distribution, which is shown as follows:

$$\tilde{c}_{ij} = c_{ij} \cdot \exp(\theta_{ij}), \ \theta_{ij} \sim \mathcal{N}(0, \sigma^2), \tag{1}$$

where $\tilde{c}_{ij}$ and c_{ij} are actual resistance and target resistance of a memristor in the i-th row and j-th column of $\boldsymbol{C}$, θ_{ij} denotes the zero-mean Gaussian variation with a variance of σ. An example of a memristive crossbar with SAFs and variations is shown in Fig. 1.

B. Problem Formulation

In this work, the computation accuracy of NCS is given by the probability that the trained NCS can successfully classify the test samples. Specifically, the higher, the better. We define the fault and variation tolerance problem in NCS as follows:

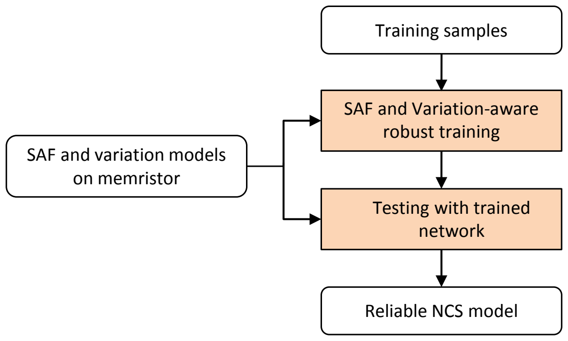

Training samples
SAF and Variation-aware robust training
SAF and variation models on memristor
Testing with trained network
Reliable NCS model

Fig. 2: The proposed reliability-driven network training for NCS.

Input: Training and testing sets, and the device variation model.
Output: A neural network model for the memristive crossbar-based NCS, which the impacts of faults and device variations are alleviated.
Objective: Improving the NCS robustness and boosting the computation accuracy of NCS.

C. Overall Flow

The overall flow of the proposed reliability-driven network training framework for NCS is shown in Fig. 2. Given the training samples and the memristor SAF and variation models, a general SAF and variation-aware robust training is performed. Through a dropout-inspired technique and a new weighted error function, the robustness of NCS to SAFs and device variations is enhanced. Then the trained neural network model is tested on test samples with Monte-Carlo simulation method to show the robustness. Finally, a reliable NCS model is achieved.

III. Reliability-Driven Network Training for NCS

In this section, we first introduce our dropout-inspired technique. Then we give the details about the new weighted error function.

A. Dropout-inspired Technique

In the general reliability-driven network training scheme, a dropout-inspired technique is proposed to prevent the complicated dependencies between weights and make the network robust against SAFs. We train the network with the mini-batch gradient descent (MGD) [16] approach, which can provide a more efficient computation process than the stochastic gradient descent (SGD). A group of instances are randomly picked from the training set in each training iteration. For each batch, we sample a network with random SAFs distribution. The weight with SA0 fault or SA1 fault is temporarily set to the maximum value or the minimum value in the current network. Note that the weights with SAFs will not participate in the back-propagation. Hence, the operation of injecting SAFs into weights is similar to the dropout strategy in deep learning [17], where the neurons are temporarily removed from the network. As a result, a weight in the network does not rely on other specific weights, and the complex co-adaptation of weights is prevented.

B. Weighted Error Function

To obtain a smooth error curve, where the effects of variations are suppressed, a new weighted error function is designed. Generally, regularization is adopted to control the complexity of the network model and avoid over-fitting. The regularization term can be expressed by adding a penalty to the error function of the learning algorithm as Equation (2):

$$\tilde{E}(\boldsymbol{W}) = E(\boldsymbol{W}) + \lambda\Omega, \tag{2}$$

Algorithm 1 General Reliability-driven Network Training

```
Input: Training set, $\boldsymbol{W}_t$ and $\eta_t$.
Output: $\boldsymbol{W}_{t+1}$ and $\eta_{t+1}$.
1: for $i \leftarrow 1$ to $T$ do
2:     Sample a mini-batch $B_i$ from training set;
3:     Inject randomly distributed SAFs into network;
4:     Calculate error $\tilde{E}(\boldsymbol{W}_t)$;                ▷ Equation (5)
5:     Obtain the accumulated gradient of error $\frac{\partial \tilde{E}(\boldsymbol{W}_t)}{\partial \boldsymbol{W}_t}$;
6:     Update $\boldsymbol{W}_t$;                                    ▷ Equation (6)
7: end for
```

where $\boldsymbol{W}$ is the weight matrix, and the parameter $\lambda > 0$ controls the relative importance of the data-dependent error $E(\boldsymbol{W})$ (typically cross-entropy error) and regularization penalty Ω.

Two popular regularizers are l_1-norm regularizer and l_2-norm regularizer as Equations (3) and (4):

$$\Omega_{l_1}(\boldsymbol{W}) = \|\boldsymbol{W}\|_1 , \tag{3}$$

$$\Omega_{l_2}(\boldsymbol{W}) = \|\boldsymbol{W}\|_2^2 . \tag{4}$$

L_1-norm regularizer has the property that if λ is sufficiently large, some weights w_{ij} are driven to zero, leading to a sparse network model. Compared with l_1-norm regularizer, l_2-norm regularizer gives more bias to the large weights during training and shrinks the weight distribution to the small value range [18]. To compensate for the impact of faults and device variations, a smooth error curve is needed. According to the work [19], l_2-norm can reduce the sensitivity of the network and thus enhance the training robustness. Besides, if the first-order derivatives of the error function with respect to weights are adopted as a regularizer, it disfavors error functions that change rapidly. Therefore, the first-order derivative regularizer helps to avoid sharp changes in error (and hence in output) with minor changes in weights. Combining Equation (2), (4) and the first-order derivative term, we can derive our joint objective function as:

$$\min_{\boldsymbol{W}} \; E(\boldsymbol{W}) + \lambda_1 \|\boldsymbol{W}\|_2^2 + \lambda_2 \left\| \boldsymbol{W} \odot \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}} \right\|_2^2 , \tag{5a}$$

where

$$E(\boldsymbol{W}) = -\sum_{r=1}^{N} \sum_{i=1}^{n} \{\tilde{y}_{r_i} \ln y_{r_i} + (1-\tilde{y}_{r_i}) \ln (1-y_{r_i})\}, \tag{5b}$$

where $\tilde{\boldsymbol{y}}_{\boldsymbol{r}} = \{\tilde{y}_{r_i}\}_{i=1}^{n}$ denotes a target vector, and $y_{r_i}(\cdot)$ refers to the i-th element of activation function output $y(\boldsymbol{x_r}, \boldsymbol{W})$. $\boldsymbol{x_r} \in \mathbb{R}^m$ is an input vector. Meanwhile, N is the total number of training data in each batch.

During training, neural weights with SAFs are fixed, while other weights can still be updated in the backward process as follows:

$$\boldsymbol{W}_{t+1} = g(\boldsymbol{W}_t, \eta_t, \frac{\partial \tilde{E}(\boldsymbol{W}_t)}{\partial \boldsymbol{W}_t}), \tag{6}$$

where $\boldsymbol{W}_t$ and η_t are the current weight and the current learning rate, and $\boldsymbol{W}_{t+1}$ denotes the updated weight. Meanwhile, $g(\cdot)$ refers to the optimizer for updating weights and learning rate.

The details of the proposed general reliability-driven network training scheme are illustrated in Algorithm 1. We use the adaptive moment estimation (Adam) optimizer [20] to train the network model. In each training iteration, we sample a mini-batch from the training set (line 2). Then the randomly distributed SAFs are injected into the current network (line 3). A feed-forward calculation is performed on each instance in the mini-batch. We calculate the error on each instance (line 4) and obtain the accumulated gradient of errors with respect to the weights (line 5). Finally, $\boldsymbol{W_t}$ is updated based on Equation (6) (line 6). The above process is terminated until satisfying the training iteration number T.

TABLE I: Original computation accuracy.

Network	Dataset	Accuracy
MLP	MNIST	92.8%
LeNet	CIFAR-10	86.2%
AlexNet	CIFAR-100	58.5%

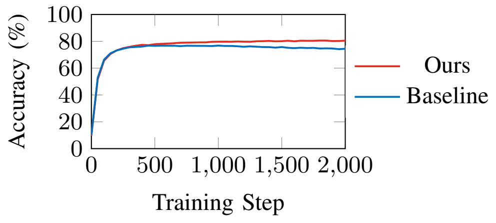


Fig. 3: Training curves.

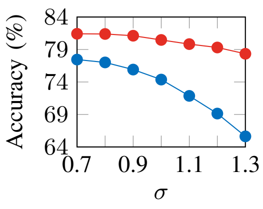
IV. Experimental Results

The framework is implemented based on `Tensorflow` library [21] and validated on a Linux server with 8-core Intel CPU and Nvidia Tesla K40M GPU. To verify the effectiveness of our algorithm, we experiment on three datasets, including MNIST [22], CIFAR-10, and CIFAR-100 [23]. First a two-layer multi-layer perceptron (MLP) is trained for MNIST. Then LeNet [22] for CIFAR-10 is implemented, which consists of two convolutional (Conv) layers and three fully connected (FC) layers. In the convolutional layers, 64 kernels of size 5×5 are employed; three FC layers are consistent, whose dimensions are 384, 192, and 10. Finally AlexNet [24] is trained for CIFAR-100. We use 3×3 kernel size to make it suitable for input images. The lognormal distribution is adopted as our memristor variation model, shown as in Equation (1). Besides, according to the results in [25], Conv layers are very sensitive to SAFs, and hence the randomly distributed SAFs are injected into the FC layers. The performance of the proposed framework is evaluated by a Monte Carlo simulation. TABLE I shows the original computation accuracy of NCS without the impacts of SAFs and variations, which serves as the upper bound of the reliability design.

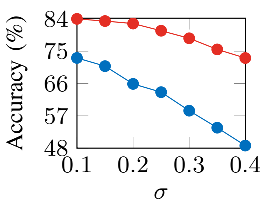
A. *Effectiveness of General Reliability-driven Network Training*

In the first experiment, we demonstrate the robustness of the proposed general reliability-driven network training to the SAFs and the device variations. The memristor variation model is injected into the weights across all different layers in the network. SAFs with 9.04% SA1 faults and 1.75% SA0 faults are added to the last FC layers [11]. We apply the proposed general reliability-driven training to train the three neural networks. For comparison, we also employ a traditional learning strategy to train the neural networks (Baseline), where the proposed dropout-inspired approach and the first-order derivative regularizer are not included. The device variation σ is set to 1. The curves of the two training methods on MNIST dataset are depicted in Fig. 3, where x-axis indicates training steps and y-axis is computation accuracy. We can see that our general reliability-driven training is a more stable training procedure and converges to a higher accuracy.

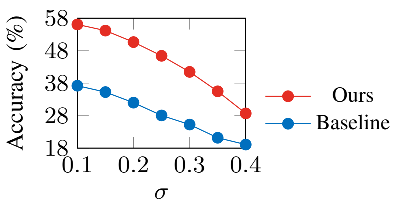
Fig. 4 illustrates the accuracy of the three neural networks derived by the two training methods, respectively. We vary σ to change the influence of stochastic device variation. It can be seen from the figure that as σ increases, the traditional learning strategy suffers from



(a) MLP on MNIST



(b) LeNet on CIFAR-10



(c) AlexNet on CIFAR-100

Fig. 4: Effectiveness of the proposed general reliability-driven training.

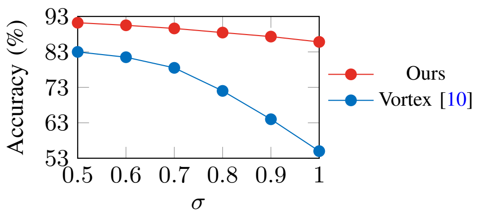


Fig. 5: Comparison between Vortex [10] and our training method.

severer accuracy degradation. Meanwhile, our general reliability-driven training can significantly improve accuracy, demonstrating the robustness to SAFs and variations. For example, our training method boosts the accuracy of MLP_MNIST from 65.63% to 78.34% even under the significant variation $\sigma = 1.3$, shown as in Fig. 4(a). A similar trend can also be observed for LeNet_CIFAR-10 and AlexNet_CIFAR-100, as shown in Fig. 4(b) and Fig. 4(c).

B. Comparison with Previous Works

In the second experiment, we compare the proposed general reliability-driven training with Vortex [10]. In Vortex, a general variation tolerant training is developed by adjusting the training goal according to the impact of the variations. The comparison is performed on MNIST dataset by using a two-layer MLP and the results are depicted in Fig. 5. The simulation results show that the proposed general reliability-driven training outperforms Vortex in terms of accuracy by 16.49% on average. Besides, a specific variation tolerant method based on mapping and re-training is presented in [11]. However, in reality, the stochastic variation value of memristors in crossbar can hardly be tested in advance.

V. Conclusion

In this paper, we have proposed a reliability-driven neural network training framework for memristive crossbar-based neuromorphic computing systems, with taking account of both SAFs and variations challenges simultaneously. Experimental results show that the proposed method can improve the computation accuracy of NCS and enhance the NCS robustness to SAFs and resistance variations.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (NSFC) under grant No. 61874102, 61732020, 61904047, Beijing Municipal Science & Technology Program under Grant No. Z181100008918013, and the Fundamental Research Funds for the Central Universities under grant No. JZ2019HGBZ0159.

References

[1] Y. Chen, H. H. Li, C. Wu, C. Song, S. Li, C. Min, H.-P. Cheng, W. Wen, and X. Liu, “Neuromorphic computing’s yesterday, today, and tomorrow–an evolutional view,” *Integration, the VLSI Journal*, 2017.
[2] W. Wen, C.-R. Wu, X. Hu, B. Liu, T.-Y. Ho, X. Li, and Y. Chen, “An EDA framework for large scale hybrid neuromorphic computing systems,” in *Proc. DAC*, 2015, pp. 1–6.
[3] Q. Xu, S. Chen, B. Yu, and F. Wu, “Memristive crossbar mapping for neuromorphic computing systems on 3D IC,” in *Proc. GLSVLSI*, 2018, pp. 451–454.
[4] S. Acciarito, A. Cristini, G. Susi, and et.al, “Hardware design of lif with latency neuron model with memristive stdp synapses,” *Integration, the VLSI Journal*, 2017.
[5] C. Song, B. Liu, W. Wen, H. Li, and Y. Chen, “A quantization-aware regularized learning method in multilevel memristor-based neuromorphic computing system,” in *Proc. NVMSA*, 2017, pp. 1–6.
[6] W. Huangfu, L. Xia, M. Cheng, X. Yin, T. Tang, B. Li, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, “Computation-oriented fault-tolerance schemes for RRAM computing systems,” in *Proc. ASPDAC*, 2017, pp. 794–799.
[7] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, “Stuck-at fault tolerance in RRAM computing systems,” *IEEE JETCAS*, vol. 8, no. 1, pp. 102–115, 2018.
[8] C. Liu, M. Hu, J. P. Strachan, and H. Li, “Rescuing memristor-based neuromorphic design with high defects,” in *Proc. DAC*, 2017, pp. 1–6.
[9] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, “Fault-tolerant training with on-line fault detection for rram-based neural computing systems,” in *Proc. DAC*, 2017, p. 33.
[10] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, “Vortex: variation-aware training for memristor x-bar,” in *Proc. DAC*, 2015, p. 15.
[11] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, “Accelerator-friendly neural-network training: learning variations and defects in RRAM crossbar,” in *Proc. DATE*, 2017, pp. 19–24.
[12] A. J. Van de Goor and Y. Zorian, “Effective march algorithms for testing single-order addressed memories,” *Journal of Electronic Testing*, vol. 5, no. 4, pp. 337–345, 1994.
[13] C.-Y. Chen, H.-C. Shih, C.-W. Wu, C.-H. Lin, P.-F. Chiu, S.-S. Sheu, and F. T. Chen, “RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme,” *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 180–190, 2014.
[14] A. Asenov, S. Kaya, and A. R. Brown, “Intrinsic parameter fluctuations in decananometer mosfets introduced by gate line edge roughness,” *IEEE TED*, vol. 50, no. 5, pp. 1254–1260, 2003.
[15] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C. J. Kim *et al.*, “Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory,” in *Proc. VLSIT*, 2012, pp. 71–72.
[16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[17] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[18] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.
[19] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, “Modifying training algorithms for improved fault tolerance,” in *Proc. IJCNN*, vol. 1, 1994, pp. 333–338.
[20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, “TensorFlow: A system for large-scale machine learning,” in *Proc. OSDI*, 2016, pp. 265–283.
[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[23] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.
[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. NIPS*, 2012, pp. 1097–1105.
[25] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and W. Yu, “Fault-tolerant training enabled by on-line fault detection for rram-based neural computing systems,” *IEEE TCAD*, vol. 38, no. 9, pp. 1611–1624, 2019.