

NEUROMORPHIC SYSTEM DESIGN AND APPLICATION

by

Beiyue Liu

B.S. in Information Engineer, Southeast University, Nanjing, China, 2011

M.S. in Electrical Engineering, University of Pittsburgh, Pittsburgh, 2014

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2016

UNIVERSITY OF PITTSBURGH
SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Beiyue Liu

It was defended on

March 30, 2016

and approved by

Yiran Chen, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Hai Li, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Xin Li, Ph.D., Associate Professor,

Department of Electrical and Computer Engineering, Carnegie Mellon University

Zhi-Hong Mao, Ph.D., Associate Professor,

Department of Electrical and Computer Engineering

Ervin Sejdic, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Dissertation Director:

Yiran Chen, Ph.D., Associate Professor, Department of Electrical and Computer Engineering

Copyright © by Beiyue Liu

2016

NEUROMORPHIC SYSTEM DESIGN AND APPLICATION

Beiyi Liu, PhD

University of Pittsburgh, 2016

With the booming of large scale data related applications, cognitive systems that leverage modern data processing technologies, e.g., machine learning and data mining, are widely used in various industry fields. These application bring challenges to conventional computer systems on both semiconductor manufacturing and computing architecture. The invention of neuromorphic computing system (NCS) is inspired by the working mechanism of human-brain. It is a promising architecture to combat the well-known memory bottleneck in Von Neumann architecture. The recent breakthrough on memristor devices and crossbar structure made an important step toward realizing a low-power, small-footprint NCS on-a-chip. However, the currently low manufacturing reliability of nano-devices and circuit level constrains, .e.g., the voltage IR-drop along metal wires and analog signal noise from the peripheral circuits, bring challenges on scalability, precision and robustness of memristor crossbar based NCS.

In this dissertation, we quantitatively analyzed the robustness of memristor crossbar based NCS when considering the device process variations, signal fluctuation and IR-drop. Based on our analysis, we will explore deep understanding on hardware training methods, e.g., on-device training and off-device training. Then, new technologies, e.g., noise-eliminating training, variation-aware training and adaptive mapping, specifically designed to improve the training quality on memristor crossbar hardware will be proposed in this dissertation. A digital

initialization step for hardware training is also introduced to reduce training time. The circuit level constrains will also limit the scalability of a single memristor crossbar, which will decrease the efficiency of implementation of NCS. We also leverage system reduction/compression techniques to reduce the required crossbar size for certain applications. Besides, running machine learning algorithms on embedded systems bring new security concerns to the service providers and the users. In this dissertation, we will first explore the security concerns by using examples from real applications. These examples will demonstrate how attackers can access confidential user data, replicate a sensitive data processing model without any access to model details and how expose some key features of training data by using the service as a normal user. Based on our understanding of these security concerns, we will use unique property of memristor device to build a secured NCS.

TABLE OF CONTENTS

PREFACE.....	XV
ACKNOWLEDGEMENTS	XVII
1.0 INTRODUCTION.....	1
1.1 MOTIVATION	1
1.1.1 Challenge 1: Training with Imperfect Hardware.....	2
1.1.2 Challenge 2: Limited System Scalability	3
1.1.3 Challenge 3: Security Concerns in Cognitive Systems	4
1.2 DISSERTATION CONTRIBUTION AND OUTLINE	5
2.0 DESIGN BASICS	7
2.1 MEMRISTOR BASICS	7
2.2 MEMRISTOR CROSSBAR.....	9
2.3 MEMRISTOR CROSSBAR BASED NCS.....	11
2.3.1 Feedforward Sensing.....	11
2.3.2 Hardware Training.....	12
2.3.2.1 Off-device training	13
2.3.2.2 On-device training.....	15
3.0 VARIATION-AWARE OFF-DEVICE TRAINING	16
3.1 IMPACT OF DEVICE VARIATION.....	16

3.2	VORTEX	18
3.2.1	Variation-aware Training (VAT).....	18
3.2.1.1	Algorithm.....	18
3.2.1.2	Variation Tolerance vs. Training Rate	21
3.2.1.3	Self-tuning and Validation	22
3.2.2	Adaptive Mapping (AMP)	24
3.2.2.1	Basic Steps of AMP	24
3.2.2.2	Greedy mapping algorithm	26
3.2.3	Integration of VAT and AMP.....	28
3.3	EXPERIMENTS	28
3.3.1	Effectiveness of AMP.....	29
3.3.2	ADC Resolution	29
3.3.3	Design Redundancy	30
3.4	SECTION SUMMARY	32
4.0	ROBUTS ON-DEVICE TRAINING WITH DIGITAL INITIALIZATION	33
4.1	NOISE-ELIMINATING ON-DEVICE TRAINING	33
4.1.1	Impacts of Device Variation and Signal Noise.....	33
4.1.2	Noise Sensitivity of On-device Training	36
4.1.3	Noise-Eliminating Training Scheme	38
4.2	DIGITAL INITIALIZATION.....	40
4.2.1	Basic Idea.....	40
4.2.2	Digitalization of Weight Matrix	41
4.3	EXPERIMENTS AND RESULTS	42

4.3.1	Noise Elimination.....	42
4.3.2	Digital-Assisted Initialization	43
4.3.3	Case study.....	46
4.4	SECTION SUMMARY	49
5.0	SCALABILITY	51
5.1	IR-DROP LIMITS SINGLE CROSSBAR SIZE.....	51
5.1.1	Impact of IR-Drop on Memristor Crossbar.....	51
5.1.2	Problem Formulation	52
5.1.2.1	Training.....	52
5.1.2.2	Sensing.....	53
5.2	SYSTEM REDUCTION	54
5.2.1	Weight Matrix Approximation.....	55
5.2.2	One-dimensional (1-D) Reduction.....	56
5.2.3	Two-dimensional (2-D) Reduction	58
5.2.4	Implementation Example.....	59
5.3	IR-DROP COMPENSATION	62
5.3.1	Sensing Compensation	62
5.3.2	Training Compensation	65
5.4	MODEL COMPRESSION	66
5.5	EXPERIMENTAL RESULTS	68
5.5.1	Training Quality	69
5.5.2	Reading Accuracy and Selection of r	71
5.5.3	Training Performance	73

5.5.4	Area.....	74
5.5.5	Robustness.....	75
5.5.5.1	Training and Testing with IR-drop.....	75
5.5.5.2	Impact of memristor/wire resistance variation	76
5.5.5.3	Tradeoff between 1-D/2-D system reduction	77
5.5.6	Model compression	78
5.6	SECTION SUMMARY	80
6.0	SECURITY APPLICATION	81
6.1	SECURITY CONCERNS IN COGNITIVE SYSTEMS.....	82
6.1.1	Test Data Privacy.....	82
6.1.2	Training Data Security.....	83
6.1.3	Model Security	86
6.2	MODEL REPLICATION ATTACK.....	86
6.2.1	Attacking Model	86
6.2.2	Demonstration.....	89
6.3	MEMRISTOR-BASED SECURED NCS	91
6.3.1	Drifting Effect	91
6.3.2	Secured NCS Design.....	92
6.4	EXPERIMENT RESULTS	93
6.4.1	Drifting vs. Degradation.....	94
6.4.2	Replication Quality.....	95
6.5	SECTION SUMMARY	96
7.0	CONCLUSION AND FUTURE WORK	97

7.1	DISSERTATION CONCLUSION.....	97
7.2	FUTURE WORK.....	99
 7.2.1	Function Generality of Memristor-based NCS.....	99
 7.2.2	Usability of Memristor-based Secured NCS	100
	BIBLIOGRAPHY	102

LIST OF TABLES

Table 1. Simulation setup	46
Table 2. Training failure rate	48
Table 3. Experiment parameters.	69
Table 4. Recall successful rate of NCS with different sizes	78

LIST OF FIGURES

Figure 1. Metal-oxide memristor [58].	8
Figure 2. Device programming [48].	9
Figure 3. Memristor crossbar [36].	10
Figure 4. Single layer neural network[56].	12
Figure 5. (a) On-device training method, (b) Off-device training method.	13
Figure 6. Impact of device variation.	17
Figure 7. Tradeoff between variation tolerance and training rate.	22
Figure 8. Self-tuning process in training.....	23
Figure 9. Adaptive mapping.	25
Figure 10. Algorithm 1.	27
Figure 11. Effectiveness of AMP.....	29
Figure 12. ADC resolution VS test rate.	30
Figure 13. Overhead vs. Test rate.	31
Figure 14. Training under memristor variation and input noise.	35
Figure 15. Training process with noise.....	36
Figure 16. Noise elimination mechanism.	38
Figure 17. Digital initialization.....	40

Figure 18. Effectiveness of noise-eliminating training.....	43
Figure 19. Comparison of convergence rate of different initialization.....	44
Figure 20. The impact of initialization on total training time.....	46
Figure 21. 3-layer network recall rate test of dynamic threshold training algorithm.	47
Figure 22. Comparisons of overall training time.....	49
Figure 23. Voltage distribution with IR-drop.	52
Figure 24. System reduction improves reliability.....	57
Figure 25. Conceptual schematics of (a) 1-D reduction (b) 2-D reduction.	60
Figure 26. Compensation for both training and sensing process.	62
Figure 27. Sensitivity analysis based compensation.....	64
Figure 28. Model compression.	67
Figure 29. Trained resistance discrepancy.....	70
Figure 30. Recall discrepancy (a) respect to r/n, (b) respect to ε	71
Figure 31. Training time comparison.....	73
Figure 32. Area cost comparison.	74
Figure 33. Recall successful rates of three NCS designs considering IR-drop.	75
Figure 34. Model compression.	79
Figure 35. Encrypted neural network.....	83
Figure 36. Reverse estimation.....	85
Figure 37. Training and replication of the learning model.	87
Figure 38. Model replication.....	90
Figure 39. Resistance change and system degradation.....	95
Figure 40. Effectiveness of memristor-based secured neuromorphic system.....	96

Figure 41. Memristor crossbar-based CNN	100
Figure 42. Ideal degradation	101

PREFACE

This dissertation is submitted in partial fulfillment of the requirements for Beiyi Liu's degree of Doctor of Philosophy in Electrical and Computer Engineering. It contains the work done from September 2011 to March 2016. My advisor is Yiran Chen, University of Pittsburgh, 2010 – present.

The work is to the best of my knowledge original, except where acknowledgement and reference are made to the previous work. There is no similar dissertation that has been submitted for any other degree at any other university.

Part of the work has been published in the conference:

1. DAC2013: **B. Liu**, M. Hu, H. Li, ZH. Mao, Y. Chen, T. Huang, W. Zhang, “Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine,” Design Automation Conference (DAC), pp. 1-6, 2013.
2. ICCAD2014: **B. Liu**, X. Li, T. Huang, Q. Wu, M. Barnell, H. Li, Y. Chen, “Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems,” International Conference on Computer-Aided Design (ICCAD), pp. 63-70, 2014.
3. DAC2015: **B Liu**, X Li, Q Wu, T Huang, H Li, Y Chen, “Vortex: variation-aware training for memristor X-bar,” Design Automation Conference (DAC), pp. 1-6, 2015.

4. DAC2015: **B Liu**, C Wu, Q Wu, M Barnell, Q Qiu, H Li, Y Chen, “Cloning your mind: security challenges in cognitive system designs and their solutions,” *invited* to Design Automation Conference (DAC), pp. 95, 2015.

Part of the work has been published in journal publications:

5. **B Liu**, Y Chen, B Wysocki, T Huang, “Reconfigurable neuromorphic computing system with memristor-based synapse design,” Neural Processing Letters, vol. 41, pp. 159-167, 2015.

ACKNOWLEDGEMENTS

I would like to acknowledge the support of my advisor, Yiran Chen, whose support made this work possible, and to 50th Design Automation Conference (DAC 2013) Richard Newton Young Student Fellow award for providing financial support. I'd like to thank Professor Yiran Chen and Professor Xin Li for their excellent guidance during the research. Professor Yiran Chen gives me guidance of emerging nonvolatile memory designs and neuromorphic system development. Professor Xin Li gives me guidance of CAD tool development, simulations and validations. Special thanks go to Professor Hai (Helen) Li, Professor Zhi-Hong Mao, and Professor Ervin Sejdic for being my committee members.

Besides, I'd like to express my gratitude to the members from Evolutional Intelligent (EI) lab at Swanson School of Engineering for their consistent supports during my research. Finally, I'd like to thank my parents for their great encouragement during the whole Ph.D. research.

1.0 INTRODUCTION

1.1 MOTIVATION

Machine learning technology has been widely used in data processing to help users better understand the underlying property of the data [1]. As a popular type of machine learning algorithm, neural network processes input data by multiplying them with layers of weighted connections. Various types of neural network designs, e.g., convolutional neural network (CNN) [2] and recurrent neural network (RNN) [3], have repeatedly and significantly improved the best performances in the literature for multiple databases from different application fields, including computer vision and nature language processing [3][4]. However, the neural network is a computation intensive software algorithm and it is a remarkable fact that implementations of a lot of milestone neural network models were enabled by the significant hardware breakthroughs, e.g., graphic processing unit (GPU)[5].

In recent years, computer hardware industry is experiencing great revolutions on its two foundation stones: semiconductor manufacturing and computing architecture: On the one hand, the scaling of conventional CMOS devices is approaching the limit [6][7]. Scalable emerging nano-devices, i.e., spintronic and resistive devices (memristor) [8]-[12], nanotube[14][15] etc., are under extensive investigations; on the other hand, the well-known “memory wall” challenge of von-Neumann architecture [8], i.e., the ever-increasing gap between CPU performance and

memory bandwidth, motivates many studies on alternative computing architectures for highly parallel software algorithms, e.g., neural network.

Neuro-biological architecture is one of such promising candidates. After twenty-year trough, neuromorphic computing, which denotes the VLSI realization of neuro-biological architecture, is recently revitalized by the discovery of nanoscale resistive devices, e.g., memristor[13]. The similarity between the programmable resistance state of memristors and the variable synaptic strengths of biological synapses dramatically simplifies the design of neural network circuits [17]-[25]. Moreover, the crossbar structure, which is the densest interconnect topology that can be achieved by modern planar semiconductor manufacturing, further boosts the integration density and power efficiency of memristor-based neuromorphic computing systems (NCS) [26]-[47] to the levels of 10^{10} -Synapses/ Inch^2 and Tera-flops/Watt, respectively. Besides, memristor crossbar structure is recently introduced to improve the execution efficiency of the Matrix-Vector multiple llications, which is one of the most common operations in the mathematic representation of neural network [37]. However, the implementation of an NCS with memristor crossbar is facing several major technical challenges mainly introduced by the physical limitations of the hardware circuit.

1.1.1 Challenge 1: Training with Imperfect Hardware

In machine learning theory, “training” is defined as the process of calculating the value of all the variables in a specific model based on training data. In memristor crossbar based NSC, we need to not only calculate the values of all variables, but also have memristors programmed to accurately represent those values. For clarification purpose, we use “hardware training” to define the whole process of calculating and device programming.

The most intuitive hardware training scheme is called the “off-device” method, which separates the whole process into two steps: The first step is identical to the conventional software training, which calculates all the variables based on the given training data. The second step is programming every memristor based on the calculation in the first step [42]. Due to the difficulty of accurate real-time monitoring the memristor state, the off-device training is vulnerable to the intrinsic device switching variations and manufacturing defects.

Surprisingly, the process of hardware training does not necessarily need to be separated into two steps. Another type of hardware training scheme is the “on-device” method, which directly implements gradient descent training (GDT) algorithm on the memristor crossbar by repeating the loop of “programming and sensing” [30]. On-device method is able to adaptively adjust the training inputs to reduce the impact of variability of memristors by sensing the memristor (indeed, output current from the crossbar) in the real-time. However, due to the very limited precision of analog signals on the hardware, the quality of on-device training is severely affected by the signal noise and sensing accuracy. At the same time, iteratively programming and sensing slows down the overall hardware training process.

1.1.2 Challenge 2: Limited System Scalability

Besides the hardware training challenges, the scale of single memristor crossbar is limited by the IR-drop along the resistance network (memristor crossbar) composed of metal wire and memristors. The analysis of the impact of IR-drop on crossbar-based digital memory shows a 64×64 crossbar already has severe voltage degradation [57]. Following the increase of the memristor crossbar size, the impact of the IR-drop becomes more critical, resulting in the performance variations or even functional failures of the NCS. Even though a large scale neural

network can be partitioned and possibly mapped onto multiple memristor crossbars, the significant hardware/energy/speed overhead of “partition & mapping” [46] makes the high single crossbar capacity a very important research challenge.

1.1.3 Challenge 3: Security Concerns in Cognitive Systems

Besides the design challenges on NCS hardware, cognitive systems, e.g., machine learning algorithm/models, that are implemented on memristor crossbar, or any hardware platforms, also have security concerns. Common cognitive systems work as the following process: Given a subset of certain type of data (training data), the cognitive system will try to extract (learn) patterns or intrinsic relationships (trained model) between variables. Then, based on the model built upon the training data, the cognitive system can make prediction/inference unknown data (test data). In this process, there are three key elements: training data, trained model and test data. There are many scenarios that one or more of these three elements are confidential or highly valuable to the system owner. Among all the security concerns, the security of model privacy interests us most. Running learning models on an embedded device introduces an obvious convenience such as run-time processing and high efficiency, but unfortunately also introduces security challenges. The learning model will be exposed to the risk of being attacked by unauthorized attackers who have physical access to the device.

1.2 DISSERTATION CONTRIBUTION AND OUTLINE

According to above three challenges, our proposed work can be also decoupled as following four main research scopes: 1) Eliminate the impact of device variation on the off-device training method; 2) Improve training quality and speed of on-device method ,which is limited by the precision and time consumption of analog computing; 3) Enhance the system scalability by reducing the required crossbar size for large network model and increasing the size of single implementable crossbar; 4) Utilizing the unique property of memristors for a learning system platform that protect model privacy against security attack.

Section 2.0 will introduce the background knowledge of memristor devices and also describes two different hardware training method in details.

Our work for research scope 1 will be described in section **3.0** . We perform an insightful analysis on the impacts of hardware design factors on the off-device training quality of NCS. Based on our analysis, we propose a novel variation-aware off-device training scheme, namely, Vortex for the training robustness enhancement: it firstly modifies the programming pre-calculation algorithm to compensate the impact of memristor variations and then introduces an adaptive mapping process to selectively map the synapse with large impact on network output onto the memristor with low variations. Integrating these two complimentary techniques together can further improve training quality.

In section **4.0** , we quantitatively analyzed the sensitivity of the on-device hardware training method to the process variations and input signal noise for research scope 2. We then proposed a noise-eliminating training method with the corresponding modified crossbar structure to minimize the noise accumulation during the on-device training and enhance the trained system performance, i.e., the testing accuracy. A digital initialization step for memristor crossbar

training is also introduced to reduce the training failure rate as well as the training time. Experimental results show that our technique can significantly improve the performance and training time of neuromorphic computing system by up to 39.35% and 23.33%, respectively.

Section 5.0 will focus on research scope 3. We will investigate the IR-drop caused physical limitation and reliability issue in memristor crossbars. More specifically, we will first formulate the effect of IR-drop in NCS designs and evaluate its impact. In order to enhance the computing capacity and reliability of NCS, we propose a system reduction scheme that can effectively reduce the required crossbar size for a specific problem while still maintaining high computation accuracy and robustness, enabling simpler and more scalable NCS implementations. To further improve the robustness of NCS, we propose a novel design method that can actively compensate the IR-drop induced signal degradations in training and computing. Note that system reduction and IR-drop compensation methods are implemented at different design levels and thus, complementary to each other. Experiment results demonstrate much smaller implementation area (i.e., 61.3% of original design circuit area) and better computing robustness (i.e., 27.0% computing accuracy improvement) of NCS after combining these two approaches.

Section 6.0 will show our work for research scope 4. We study the learning process that allows the attacker to attach the privacy of a model on an embedded device. We then will investigate using unique drifting property of memristor device to build a secured NCS that prevents replicating the model hard-coded in a memristor crossbars. The performance of the secured system will gradually degrade without regular calibrations.

2.0 DESIGN BASICS

2.1 MEMRISTOR BASICS

As predicted by Prof. Leon Chua in 1971 [13], memristor is the fourth fundamental circuit element uniquely defining the relationship between magnetic flux (ϕ) and electrical charge (q) as: $d\phi = M dq$. Here the electrical property of the memristor is represented by memristance (M) in unit of Ω . Since ϕ and q are time dependent parameters, the instantaneous resistance (memristance) of a memristor is determined by the historical profile of the electrical excitations through the device. In other words, the resistance state of a memristor can be programmed by applying current or voltage. In 2008, HP Labs reported that the memristive effect was realized by moving the doping front along a TiO_2 thin-film device [58]. Since then, many different memristive materials and structures were found or rediscovered [48].

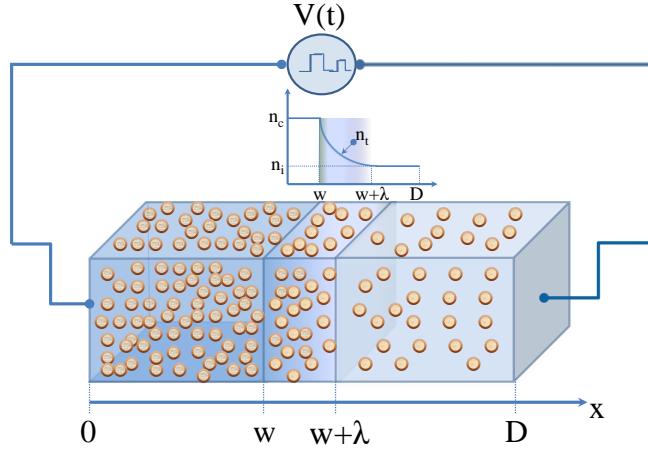


Figure 1. Metal-oxide memristor [58].

Figure 1 depicts an ion migration filament model of metal-oxide memristors [58]. A metal-oxide layer is sandwiched between two metal electrodes. During *reset* process, the memristor switches from low resistance state (*LRS*) to high resistance state (*HRS*). The oxygen ions migrate from the electrode/oxide interface and re-combine with the oxygen vacancies. A partially ruptured conductive filament region with a high resistance per unit length (R_{off}) is formed on the left of the conductive filament region with a low resistance per unit length (R_{on}). During *set* process, the memristor switches from *HRS* to *LRS*. The ruptured conductive filament region shrinks. The resistance of a memristor can be programmed to any arbitrary value between *LRS* and *HRS* by applying a programming current or voltage with different pulse widths or magnitudes. Note that the relationship between the programming voltage amplitude/pulse width and the memristor resistance change is usually a highly nonlinear function, as shown in Figure 2[48]. For example, with programming voltage of -2.9V, it takes ~500 ns to switch the device

from LRS to $900\text{ k}\Omega$ (point ‘A’ in the Figure 2). However, with the same programming time, - 2.8V only switches the device to $\sim 400\text{ k}\Omega$, which is half of the resistance marked by point ‘A’.

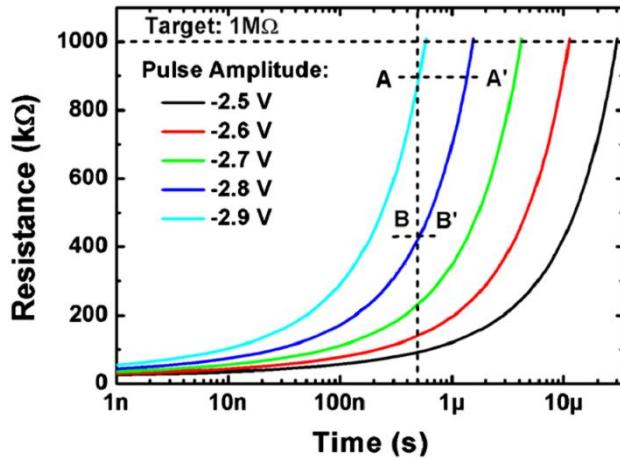


Figure 2. Device programming [48].

2.2 MEMRISTOR CROSSBAR

As shown in Figure 3, a memristor crossbar is a connection structure that integrates a matrix of memristors (M) with metal wires. Each memristor is connected to a top horizontal metal wire and a vertical bottom wire. The crossbar structure realizes the highest possible integration density of memristor devices within a single layer, in which each memristor uses $4f^2$ circuit area (f =feature size).

Read: The resistances of memristors in a crossbar can be read individually. For example, when reading the resistance of m_{ij} , which is the memristor connecting to the i -th top metal wire

and the j -th bottom metal wire, a sensing voltage v will be applied on the i -th top wire while all the other wires are grounded. The current c_j can be sensed from the j -th metal wire and resistance of $m_{ij} = v/c_j$. Besides, the resistances of a column of memristors can be sensed together as we will describe in 2.3.1.

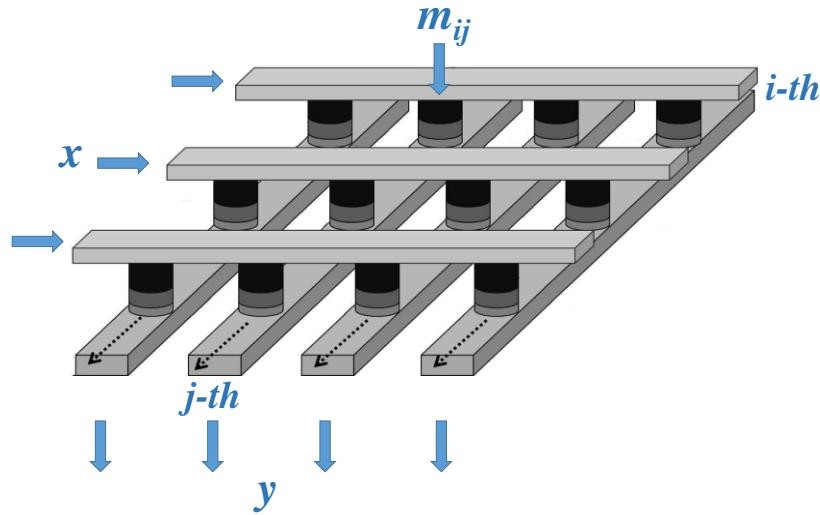


Figure 3. Memristor crossbar [36].

Program: At the same time, memristors in a crossbar can be programmed individually. During the programming of memristor crossbar, different amplitude and duration of programming pulses are directly applied to the target memristor based on the desired resistance change: the voltages of the WL and BL connecting the target memristor are set to $+V_{bias}$ and GND , respectively, while all other word-lines (WLs) and bit-lines (BLs) are connected to

$+V_{bias}/2$. Hence, only the target memristor is applied with the full V_{bias} above the threshold that can change the device's resistance state while the rest of memristors in the crossbar remain unchanged because they are only half selected with a voltage of $V_{bias}/2$ [57]. Due to the intrinsic switching characteristics, memristors with half-programming voltage barely changes their resistances.

2.3 MEMRISTOR CROSSBAR BASED NCS

2.3.1 Feedforward Sensing

Figure 4 depicts a conceptual overview of a neural network that can be implemented with a memristor crossbar based NCS in Figure 3. Two groups of neurons are connected by a set of synapses. The input neurons send signals into the network and the output neurons collect the information from the input neurons through the synapses and process them with an activation function. The synapses apply different weights (synaptic strengths) on the information during the transmission. In general, the relationship between the input pattern x and the output pattern y can be described as [28]:

$$y_n = W_{n \times m} \cdot x_m \quad (1)$$

Here the weight matrix $W_{n \times m}$ denotes the synaptic strengths between the two neuron groups. In an NCS, the matrix-vector multiplication shown in the equation (1) is one of the most intensive operations. Because of the structural similarity, a memristor crossbar is conceptually efficient in executing matrix-vector multiplications [37].

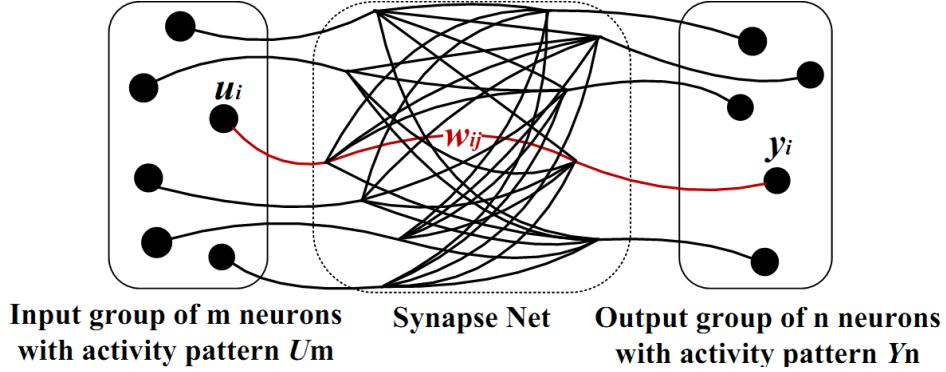


Figure 4. Single layer neural network[56].

The computation process defined by equation (1) is called “sensing”. In hardware implementation shown in Figure 3, during the sensing process of a memristor crossbar-based NCS, \mathbf{x} is mimicked by the input voltage vector applied to the WLs of the memristor crossbar while the BLs are grounded. Each memristor is programmed to a resistance state representing the weight of the correspondent synapse. The current along each BL of the memristor crossbar is collected and converted to the output voltage vector \mathbf{y} by “neurons”, e.g., CMOS analog circuit or emerging domain wall devices [36]. The matrix $\mathbf{W}_{n \times m}$ is often implemented by two crossbars, which represent the positive and negative elements of $\mathbf{W}_{n \times m}$, respectively.

2.3.2 Hardware Training

As mentioned in section 1.1.1, we use “hardware training” to define the process of calculating and programming all the memristors to the target resistant states.

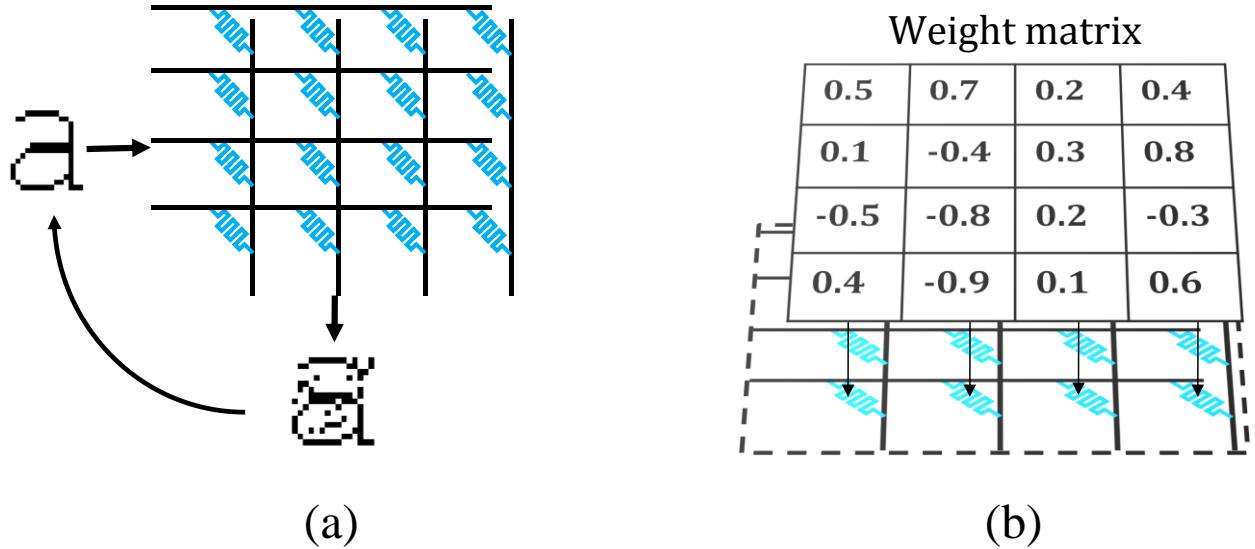


Figure 5. (a) On-device training method, (b) Off-device training method.

2.3.2.1 Off-device training

As mentioned in section 1.1.1, the most intuitive hardware-training scheme is called the “off-device” training method (Figure 5 (b)), which separates the whole process into two steps. The first step is identical to the conventional software training, which calculates all the variables. A neural network is usually trained with supervised learning method to perform a classification or regression function. Without losing generality, we use classification tasks as examples in this dissertation. Assuming we have a data set, which consists of training samples and testing samples. Each training/testing sample contains a set of feature vectors \mathbf{FR}/\mathbf{FT} and a set of label vectors \mathbf{LR}/\mathbf{LT} . The task of the neural network is to make predictions as close as \mathbf{LT} based on \mathbf{FT} .

For generality purpose, e.g., multi-task multi-class problems, we assume labels \mathbf{LR}/\mathbf{LT} as a vector of values instead of single value. Each column of memristors are used to perform

classification on one bit of labels. Hence, every column of memristors can be trained individually. For each column, the difference between the current neural network output and the training labels can be described as the following cost functions:

$$c = \sum_{s=1}^n (lr_s - o_s)^2 \quad (2)$$

Here, c is the cost value, lr_s is the s -th label in the training label vector \mathbf{LR} and o_s is the output when given the s -th training feature in \mathbf{FR} , i.e., fr_s . Assuming there are n samples in the training data, the goal of software training is minimizing the sum of error, i.e., cost value defined by equation (2). Since we are able to compare the network's calculated values for the output nodes to these "correct" labels, equation (2) can be optimized by GDT algorithm:

$$\Delta w_{ij} = \frac{\partial c}{\partial w_{ij}} = \sum_{s=1}^n (lr_s - o_s) \cdot \frac{\partial o_s}{\partial w_{ij}} \cdot \alpha \quad (3)$$

Here α is the training speed parameter and the error terms will be used to adjust the weight so that in the next time around the output values will be closer to the "correct" values \mathbf{LR} . Usually the labels are a vector of +1/-1, indicating one fr_s belongs to a certain class or not.

Once the training converges, e.g., cost value stabilizes below certain threshold, weight matrix \mathbf{W} can be used for testing. Based on \mathbf{W} , programming pulse voltage amplitude and duration for each memristor can be calculated based on the relationship shown in Figure 2. Then the memristor devices can be updated accordingly, which is the second step of off-device training.

2.3.2.2 On-device training

Surprisingly, the processes of calculating the variable values and memristor programming are not necessary to be separated. Another type of hardware training scheme is the “on-device” method (Figure 5 (a)), which directly implements GDT algorithm on the memristor crossbar by repeating the loop of “programming and sensing” [30]. On-device method is able to adaptively adjust the training inputs to reduce the impact of variability of memristors by sensing the memristor (indeed, output current from the crossbar) in the real-time.

One significant difference between our on-device training scheme and the conventional software training is the feedforward operation is done on the memristor crossbar devices. The features are applied as voltages on the input terminals and output results are sensed as introduced in section 2.3.1. There are clear benefits of implementing feedforward operation on device. Since there is no such step of programming memristors based on the pre-calculated connection weights, there is no discrepancy between theoretical weights and hardware weights. Besides, the device variation and defects can be automatically compensated by the on-device training because of the close-loop training algorithm. The main shortcoming of this training scheme is that the analog values need to be quantized through the interfaces as:

$$o_{digital} = \text{floor}(2^{bits} \cdot o_{analog})/2^{bits} \quad (4)$$

Limited by ADC hardware, the parameter *bits* cannot reach 32 or even 64 as people use it software training.

3.0 VARIATION-AWARE OFF-DEVICE TRAINING

3.1 IMPACT OF DEVICE VARIATION

As mentioned in section 1.1.1, the off-device training method separates the hardware training process into two steps, e.g., calculating weights of all connections in software and programming each memristor based on the calculation. In hardware implementation, off-device training is subject to many realistic factors and constraints. In this section, we will investigate the impacts of these limitations on the robustness of different hardware training methods. Here, the “robustness” is quantitatively measured as the test accuracy of a memristor crossbar-based NCS trained by a specific method.

The main difference between on-device and off-device training is that on-device training adaptively adjusts the programming signal during the iterations based on the sensed output current of the memristor crossbar. Theoretically, memristor device variations can be naturally tolerated in this process if the analog output current can be precisely sensed. On the contrary, the off-device training determines the programming pulse width and magnitude before accessing the devices. Hence, device variations inevitably incur the discrepancy between the targeted value and the actual programmed memristor resistance.

To illustrate the impact of device variations on the training of memristor crossbars, we performed on-device and off-device training on a column of 100 memristors. The nominal on-

and off-state resistances of the memristor are set to $10\text{ k}\Omega$ and $1\text{ M}\Omega$, respectively. Here we assume the memristor device variation follows a lognormal distribution [63]. It means that for an on-state memristor, its resistance $r \rightarrow e^{\theta} \cdot 10\text{ k}\Omega$, where $\theta \sim N(0, \sigma^2)$. The training goal is to ensure when the input wires are all connected to 1V, the memristor column shall generate an output current of 1mA. Figure 6 shows $1000 \times$ Monte-Carlo simulation results when the standard deviation σ changes. Following the increase of σ , on-device training result constantly maintains a low discrepancy between the trained output and the target output while this output discrepancy keeps growing in off-device training result. This experiment is performed by assuming the analog sensing of on-device training is perfectly precise, which is unrealistic in hardware.

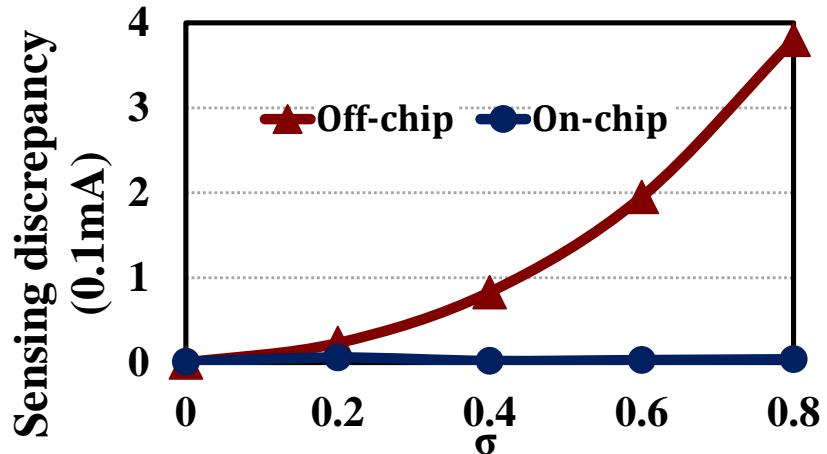


Figure 6. Impact of device variation.

3.2 VORTEX

The low requirement of sensing resolution makes off-device training an attractive solution in memristor crossbar-based NCS designs. However, compared with on-device training, the disadvantage of off-device training is also obvious: open-loop scheme is intrinsically lack of the mechanism to tolerate memristor device variations. In this work, we propose Vortex – a variation-aware robust training scheme that can tolerate the memristor device variations using the following two techniques:

- Variation-aware training (VAT) – an off-device training method that models the device variations and adjusts the training goal to tolerate the variation impact in pre-calculation step.
- Adaptive mapping (AMP) – a method to pre-test all devices and then adaptively map the synaptic connections to physical devices based on the actual memristors' variations.

3.2.1 Variation-aware Training (VAT)

3.2.1.1 Algorithm

Without losing generality, we use a one layer a neural network as an example. The goal of conventional GDT algorithm is to find the connection weights \mathbf{W} that can successfully classify the training samples as many as possible. The computation of the network can be expressed as equation (1). Here \mathbf{x} is a $1 \times n$ input feature vector, \mathbf{W} is a $n \times m$ weight connection matrix and \mathbf{y} is a $1 \times m$ output vector that corresponds to m classes. As each column of \mathbf{W} is trained separately, the training of the r -th column (\mathbf{w}_r) can be summarized as the following optimization process:

$$\begin{aligned} \text{Min} \quad & \sum_i^s \varepsilon^{(i)} \\ \text{S.T.} \quad & \widehat{y}_r^{(i)} \cdot (\mathbf{x}^{(i)} \cdot \mathbf{w}_r) \geq 1 - \varepsilon^{(i)}; \quad \varepsilon^{(i)} \geq 0. \end{aligned} \tag{5}$$

Here the i -th training sample contains input feature vector $\mathbf{x}^{(i)}$ and target output $\widehat{y}_r^{(i)}$ ($\widehat{y}_r^{(i)} \in \{-1, 1\}$). s is the total number of the training samples. The optimization process minimizes the difference between the actual output ($\mathbf{x}^{(i)} \cdot \mathbf{w}_r$) and the target output ($\widehat{y}_r^{(i)}$), given the condition that the output current of a crossbar is physically bounded by circuit limitations, which is numerically represented as “1” in equation (5). Here we use “1 vs. all” method in the output neuron design: $\widehat{y}_r^{(i)} = 1$ only when a training sample is labeled as class r .

In a memristor crossbar-based NCS, weight matrix \mathbf{W} is represented by the crossbar. When memristor variations are taken into account, the actual programmed weight matrix \mathbf{W}' may be different from the target \mathbf{W} even we can perfectly control the programming voltage pulse width and magnitude. Similar to Section 3.1, here we assume the memristor device variation follows lognormal distribution [63] as $\mathbf{w}_r' \rightarrow \text{diag}(e^\theta) \cdot \mathbf{w}_r$, and $\theta_i \sim N(0, \sigma^2)$, $i = 1 \dots n$. The optimization constraint of equation (5) then becomes:

$$\widehat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{rq} \cdot e^{\theta_q} \geq 1 - \varepsilon^{(i)}. \tag{6}$$

As θ is a small variation, we can simplify the equation (6) using a linear approximation:

$$\widehat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{rq} \cdot (\alpha_0 + \alpha_1 \cdot \theta_q) \geq 1 - \varepsilon^{(i)}. \tag{7}$$

Equation (7) can be further reorganized as:

$$\frac{\text{penalty of variation}}{\alpha_1 \cdot \hat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{r_q} \cdot \theta_q} + \frac{\text{conventional training}}{\alpha_0 \cdot \hat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{r_q}} \geq 1 - \varepsilon^{(i)}. \quad (8)$$

The second term of equation (8) is also used as the constraint in the conventional GDT algorithm. We call the first term as “penalty of variations” because it represents the sum of the crossbar output deviation induced by device variations. However, the optimization process cannot be performed with random variable θ_q . Therefore, we estimate the upper bound of the penalty of variations by:

$$\sum_{q=1}^n x_q^{(i)} \cdot w_{r_q} \cdot \theta_q \leq \|\theta\|_2 \cdot \|V^{(i)}\|_2 \quad (9)$$

Here, $V^{(i)} = [\mathbf{w}_r \cdot x_1^{(i)} \cdots \mathbf{w}_r \cdot x_n^{(i)}]^T$, $\theta = [\theta_1 \cdots \theta_n]$.

$\|\theta\|_2$ is the 2-norm of a vector of random variables that follow normal distributions. At a certain confidence level, we can restrict $\|\theta\|_2 \leq \rho$ based on Chi-square distribution with degree of freedom n . Then the modified training process under the consideration of weight variations can be expressed as:

$$\begin{aligned} \text{Min} \quad & \sum_i^s \varepsilon^{(i)} \\ \text{S.T.} \quad & \left\{ \begin{array}{l} V^{(i)} = [x_1^{(i)} \cdot \mathbf{w}_r \cdots x_n^{(i)} \cdot \mathbf{w}_r]^T \\ t^{(i)} \geq |V^{(i)}| \\ \rho \cdot |\alpha_1 \cdot \hat{y}_r^{(i)}| \cdot t^{(i)} - \alpha_0 \cdot \hat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{r_q} + 1 - \varepsilon^{(i)} \leq 0 \\ \varepsilon^{(i)} \geq 0 \end{array} \right. \end{aligned} \quad (10)$$

We refer to this technique as VAT (variation-aware training).

3.2.1.2 Variation Tolerance vs. Training Rate

The introduction of the estimated penalty of variations makes the training procedure be aware of device variations at a predetermined degree and include them in the training constraints. The trained memristor crossbar, hence, becomes more robust in tolerating device variations during computations, allowing us to obtain the desired output even there are variations in the programmed weights. However, such a method applies a tighter constraint to the training and results in lower training rate.

To evaluate the tradeoff between the training rate and the variation tolerance of the NCS under VAT, we vary the estimated penalty of variation in equation (10) by multiplying a scalar γ ($0 < \gamma < 1$):

$$\gamma \cdot \rho \cdot |\alpha_1 \cdot \hat{y}_r^{(i)}| \cdot t^{(i)} - \alpha_0 \cdot \hat{y}_r^{(i)} \cdot \sum_{q=1}^n x_q^{(i)} \cdot w_{r_q} + 1 - \varepsilon^{(i)} \leq 0, \quad (11)$$

and simulate the corresponding training rate and test rate. When γ changes from 0% to 100% (none to the maximum estimated penalty of variations), the training rate keeps reducing, as shown in Figure 7.

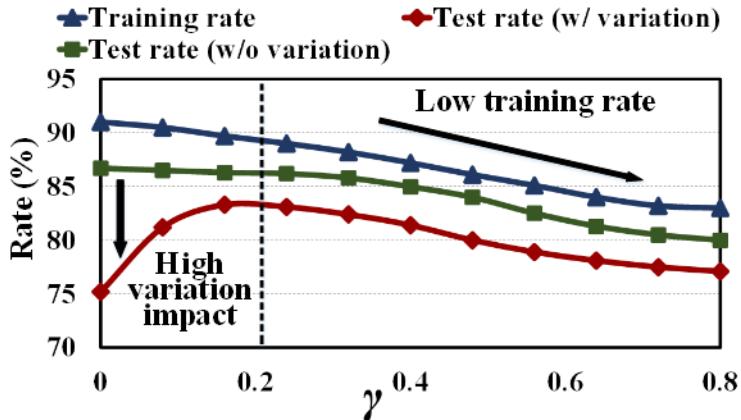


Figure 7. Tradeoff between variation tolerance and training rate.

The left side of Figure 7 shows test rate (w/ variation) is significantly lower than test rate (w/o variation), indicating significant impact of device variations on the training quality in this range. When γ rises, test rate (w/o variation) continues to decreases due to the disturbance of the introduced estimated penalty of variations to the optimization process. Teste rate (w/ variation), however, raises to a peak first when γ increases to 0.2. It clearly shows the efficacy of VAT to tolerate device variations in the training. Continue increasing γ , however, may not further improve the variation tolerance. The disturbance to the training process starts to dominate and results in a decrease of the test rate.

3.2.1.3 Self-tuning and Validation

Figure 7 shows that for a specific memristor crossbar based NCS, there exists an optimal γ that ensures the maximum test rate (but not corresponding to the highest training rate). Hence, we propose a self-tuning process that is very similar to the regularization used in regressions to prevent over-fitting and maximize test rate [67]. The details of the proposed process are shown in

Figure 8. Instead of training the neural network only once with all training samples, we separate the training samples into two groups (one is large and one is small). The large group is used as the actual “training samples” while the small group is used for “validation”. After training, a validation step will be launched: we first model the memristor variations and inject them into the weight matrix W trained by the training samples. Then the training quality of the NCS is tested by the validation samples under a fixed γ . We repeat training-validation loops by scanning the value of γ until achieving the maximum test rate over the all validation samples and the corresponding γ will be selected in the final training process. Note that the efficacy of self-tuning loop varies when the memristor device variation model changes. In this paper, we use the lognormal distribution as our memristor device variation model [63]. However, our proposed techniques are not restricted to any particular variation models.

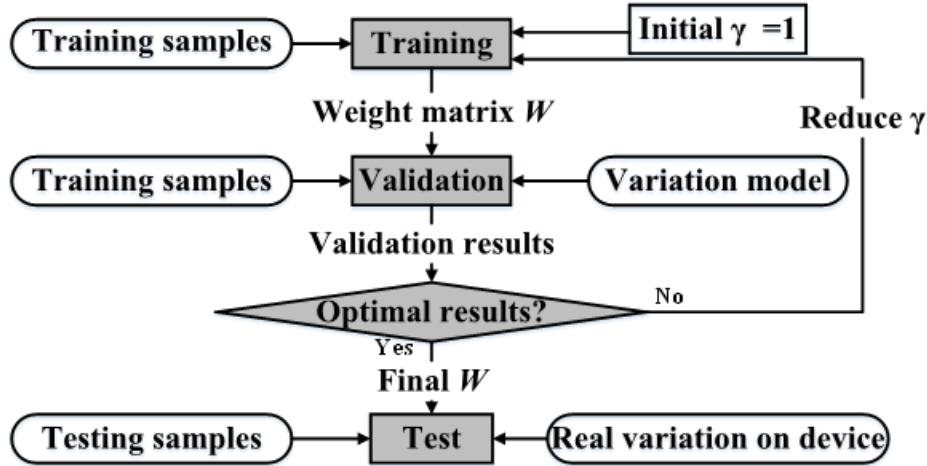


Figure 8. Self-tuning process in training.

3.2.2 Adaptive Mapping (AMP)

VAT aims optimizing the training algorithm to tolerate the impact of device variations. In this section, we propose adaptive mapping (AMP) – a hardware solution to mitigate the impact of device variations by optimizing the mapping scheme of the computation to the crossbar and leveraging the design redundancy.

3.2.2.1 Basic Steps of AMP

AMP includes three sequential steps:

Pre-testing – After a memristor crossbar is manufactured, we program every memristor targeting a certain resistance state and then sense the device resistance to get the distributions of memristor resistance in a the crossbar (we may need to sense multiple times eliminate the impacts of switching variations). To minimize the impact of IR-drop and sneak paths, we would perform pre-testing on each individual memristor and keep all other memristors at high-resistance state (HRS). The obtained distribution should follow lognormal distribution [63].

Sensitivity analysis – Variability of different memristors generates different impact on the computation accuracy of a NCS. To identify the memristors that have large impacts on the NCS computation accuracy and need better control of device variations, a sensitivity analysis is performed. In an $m \times n$ crossbar, the sensitivity of the j -th output y_j to the device variation of a specific memristor ($e^{\theta_{ij}}$) is:

$$\frac{\partial y_j}{\partial e^{\theta_{ij}}} = \frac{\partial (\sum_{i=1}^n x_i \cdot (w_{ij} \cdot e^{\theta_{ij}}))}{\partial e^{\theta_{ij}}} = x_i \cdot w_{ij} \quad (12)$$

Equation (12) shows that the impact of a memristor's variations on the NCS computation accuracy is proportional to the product of the input and the weight that memristor represents. Since the weight matrix \mathbf{W} of a neural network is often highly skewed (e.g., $\max(w_{ij})$ is easily $>1000 \times \min(w_{ij})$), the memristors with a low resistance and a high input demand for a better variation control.

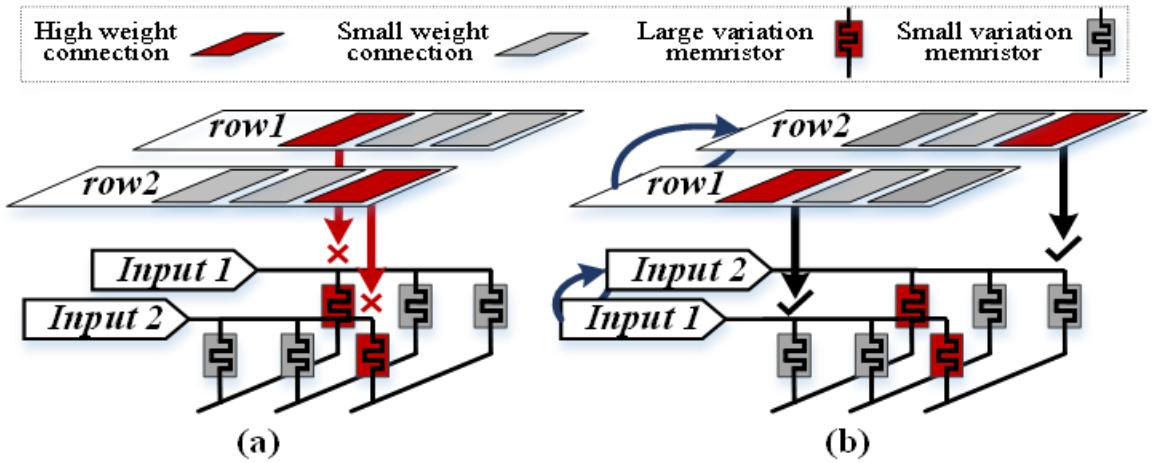


Figure 9. Adaptive mapping.

Mapping – To minimize the impact of device variations on the NCS computation accuracy, we may replace the memristor with a resistance significantly deviating from the nominal value and high impact on NCS computation accuracy using a device with smaller variation. But it is hard to physically replace a memristor with another after a crossbar being fabricated. However, changing the mapping relation between elements in the weight matrix and memristors in the crossbar can be easily done. In observation of the multiplication of

permutations of vector-matrix multiplication, switching two rows in weight matrix together with their inputs does not change the output of the multiplication. Hence, if one row (e.g., row1 in Figure 9) in the crossbar has one memristor with large variation that matches a high weight connection, we can assign the input signals originally on row1 to the input of another row (e.g., row2 in Figure 9) and program row2 to the original weight of row1. In this case, all high weight connections and large variation memristors have been mismatched.

3.2.2.2 Greedy mapping algorithm

To achieve a minimum impact of memristor device variations across the whole cross, we adopt a greedy mapping algorithm in AMP to determine the mapping relations between the weight matrix and the crossbar, as depicted in Figure 10. The whole mapping process can be summarized as the follows:

We first calculate the impact of device variations by mapping the p -th row of weights matrix onto the q -th row of the memristor crossbar. As discussed in sensitivity analysis, such an impact can be measured by “summed weighted variations (SWV)” as:

$$SWV_{pq} = \sum_{j=1}^n |w_{pj} \cdot (e^0 - e^{\theta_{qj}})| = \sum_{j=1}^n |w_{pj} \cdot (1 - e^{\theta_{qj}})|. \quad (13)$$

Here we assume both the crossbar and weight matrix \mathbf{W} have total n columns. w_{pj} is a connection weight at the location (p,j) in \mathbf{W} and $e^{\theta_{qj}}$ represents the device variation of the memristor at the location (q,j) in the crossbar. $w_{pj} \cdot (e^0 - e^{\theta_{qj}})$ show the difference between the ideal weight ($w_{pj} \cdot e^0$) and the actual weight represented by the memristor ($(1 - e^{\theta_{qj}})$). Here θ $\sim N(0, \sigma^2)$.

Algorithm 1: Greedy mapping algorithm

```

1 Xbar = abs(Ones(m×n) - Xbar_variation (m×n));
2 For (row#=1, row#<m, row#++){
3     Target_row# = 1;
4     Min_SWV = abs (Weight (row#,:) × Xbar (1,:)');
5     For(scanner=1, scanner<n, scanner#++){
6         If Min_SWV > abs (Weight (row#,:) × Xbar (scanner,:)');
7             Target_row#=scanner#;
8         End
9     End
10    Pair(row#) = Target_row#;
11    Remove Xbar (Target_row#);
12 End

```

Figure 10. Algorithm 1.

The mapping starts with the row of \mathbf{W} with the largest device variation sensitivity calculated in equation (12) and maps it to the row of the crossbar with the smallest SWV. After a row is mapped, its original row from \mathbf{W} and the mapped row in the crossbar will be removed from the queue of the to-be-mapped rows. AMP will repeat this process until all the rows are properly mapped. As many redundant designs, we may also leverage additional memristor columns/rows to further improve the efficacy of the mapping. The mapping algorithm remains almost the same expect that there are more memristor rows are available.

Defective cell is another reliability issue in the fabrication of memristor crossbars, causing the device resistance at HRS or LRS. Such defective cells can be detected as memristors with large variations and replaced by AMP by following the similar approach.

3.2.3 Integration of VAT and AMP

VAT and AMP are two complementary techniques that can be seamlessly integrated while the efficacies of them are also stackable: For example, if effective device variations of the memristor crossbar have been reduced by AMP, this reduction shall be captured by the memristor device variation model used in the self-tuning process of VAT. As a result, a smaller penalty of variation will be needed in VAT, leading to potentially higher training rate and test rate.

3.3 EXPERIMENTS

To evaluate our proposed *Vortex* scheme, we implement a two-layer neural network on a memristor crossbar based NCS for the famous MNIST digits classification task [76]. The input signals of the crossbars are digital voltages corresponding to the pixels of the original benchmark images. The output signals are the currents sensed from the ten vertical wires of the crossbar; each of them represents one class from ‘0’ to ‘9’. “1 vs. all” method is still used in output neuron designs. Each benchmark image has 28×28 pixels, requiring a 784×10 crossbar for the computation. The nominal on-state and off-state resistances of memristors used in our experiment are $10\text{k}\Omega/1\text{M}\Omega$ respectively. Benchmark may need to be under-sampled to fit into the memristor crossbars with difference sizes in the relevant evaluations.

3.3.1 Effectiveness of AMP

Figure 11 illustrates the training rate of VAT and test rates of the crossbar before and after applying AMP. As expected, after AMP is applied, the impact of device variations of the crossbar decrease, resulting in improvement of test rate w.r.t. the case before AMP is applied. Besides, the optimal γ also reduces from 0.4 (before AMP) to 0.2 (after AMP).

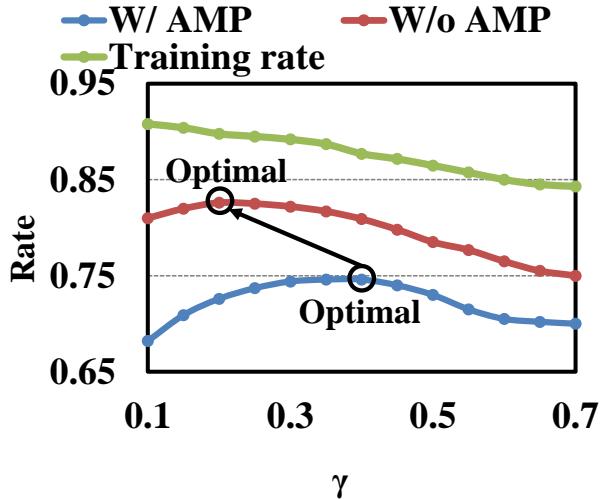


Figure 11. Effectiveness of AMP.

3.3.2 ADC Resolution

The resolution of analog-digital converter (ADC) is an important factor that affects the efficacy of AMP by influencing the memristor resistance pre-testing accuracy. We analyze the impact of different resolutions on NCS computation robustness (test rate). No redundancy is added in this analysis. Figure 12 shows the test rates of the NCS with different ADC resolutions

under different device variations. Low resolution (4-bit/5-bit) significantly limits the computation robustness. The test rates of the NCS with different variations start to saturate when a 6-bit ADC is applied. Further improving the ADC resolution gives us very marginal computation robustness enhancement. Hence, we fix the ADC resolution at 6-bit in the following experiments.

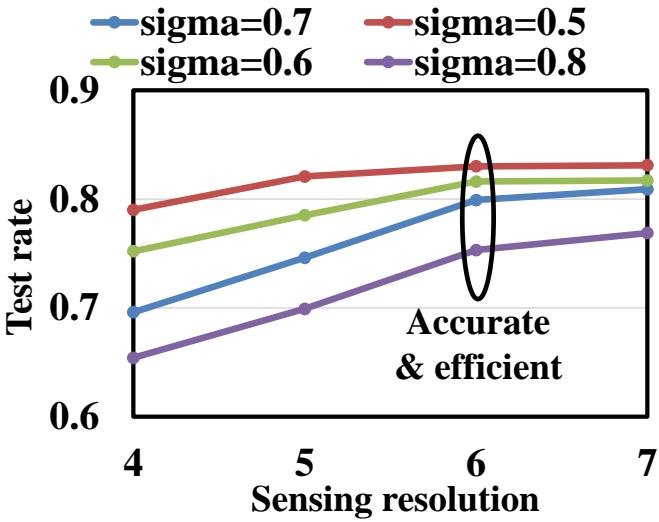


Figure 12. ADC resolution VS test rate.

3.3.3 Design Redundancy

We analyze the tradeoff between design redundancy and NCS computation robustness using the same experiment setup in Section 3.3.1. When memristors have a large variation ($\sigma=0.8$) and there are no redundant rows, the test rate is generally low, i.e., 71.8%. To improve the computation robustness, we may add extra p rows. Figure 13 shows the test rates of the crossbar with different p under different training schemes.

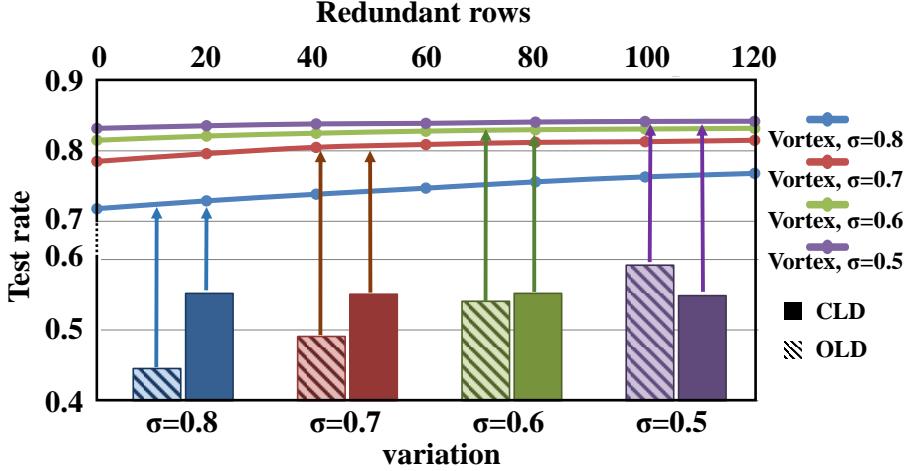


Figure 13. Overhead vs. Test rate.

In general, increasing the redundancy (p) helps to improve the test rates. However, the test rates are primarily determined by the device variations rather than the redundancy, and the help of redundancy is more prominent when the device variations are large. For comparison purpose, Figure 13 also shows the test rates under conventional off-device and on-device training without design redundancy. On average, Vortex achieves 29.6% and 26.4% higher test rates compared to off-device and on-device training, respectively, even without redundant rows. Here the theoretical maximum test rate in this configuration is ~85%, which is determined by the nature of the adopted neural network model. In the following experiments, we choose 100 redundant rows and $\sigma=0.6$ as our default setup.

3.4 SECTION SUMMARY

In this section, we try to understand the training robustness of memristor crossbars by quantitatively analyzing the influences of some hardware limitations, e.g., device variation, and sensing resolution. Based on our analysis, “Vortex” – a variation-aware off-device training scheme is then developed to better tolerate device imperfections and design constraints. Experiment results show that Vortex achieves significantly improved training quality, i.e., 29.6% higher test rate, w.r.t. conventional off-device training.

4.0 ROBUTS ON-DEVICE TRAINING WITH DIGITAL INITIALIZATION

4.1 NOISE-ELIMINATING ON-DEVICE TRAINING

As mentioned in section 2.3.2.2, the memristor crossbar hardware training does not necessarily be separated into two steps as described in off-device training. Another type of hardware training scheme is the “on-device” method, which directly implements gradient descent training (GDT) algorithm on the memristor crossbar by repeating the loop of “programming and sensing” [30]. Since the on-device method is a close-loop operations, it may be able to adaptively adjust the training inputs to reduce the impact of variability of memristors by sensing the memristor (indeed, output current from the crossbar) in the real-time. However, the impact of the very limited precision of analog signals on the hardware on the quality of on-device training needs further investigation.

4.1.1 Impacts of Device Variation and Signal Noise

To have an intuitive view on the impact of device variation and signal noise on crossbar training, we perform the following experiment. Figure 14 shows an example of the output comparison step in the on-device training process when a set of read voltage V_{rd} , 0, $V_{rd}/2$ is applied to the WLs of three memristors R1-R3 in the same column. Here we assume the three memristors are all at HRS. The ideal voltage on the BL shared by these three memristors should

be $V_{rd}/2$. However, the device non-uniformity and the input voltage fluctuation may cause the bias changes on the memristors. For example, if the resistance of R1 is larger than that of R2, the voltage on the BL will be below $V_{rd}/2$, as shown in Figure 14 (a). Also, if the input voltages on the WL of R1 changes to $V_{rd} + \Delta V$, the voltage on the BL will be above $V_{rd}/2$, as shown in Figure 14 (b). In both cases, the calculated difference between the current output and the target output will be different from the ideal case. Such deviation can be accumulated along with the training iterations. Together with the fluctuations of the programming voltage and the process variations, it will cause the deviation of the programmed memristor resistance from the ideal value during the programming step in the memristor crossbar training process and finally affect the computation accuracy. We use an example to illustrate the impacts of the process variation and input signal noise on the memristor crossbar training. A 64 x 64 crossbar is implemented to realize the synapse connection of a one-layer neural network. Figure 14 (d) shows the resistance difference between the ideally trained crossbar (no process variation or input signal) and the crossbars trained with considering process variation (top row) or input signal noise (bottom row), respectively. In the evaluation of process variation's impact, the distribution of the memristor cell size in the crossbar is generated randomly for every iteration with Gaussian distribution. Note that since the input noise for write will result in the variation of the crossbar memristance, we consider the write input noise with process variation together. The standard deviation of the memristance variation is assumed to be 10% ($\sigma = 0.1$), 20% ($\sigma = 0.2$), and 30% ($\sigma = 0.3$) of its nominal value. In the evaluation of the read input signal noise's impact, similarly, a random noise following Gaussian distribution is generated on the input signals of the crossbar in every iteration. The standard deviation of the noise is assumed to be 10% ($\sigma = 0.05$), 20% ($\sigma = 0.1$),

and 30% ($\sigma = 0.15$) of V_{rd} . The mean of the noise is zero. The GDT rule is applied in the training.

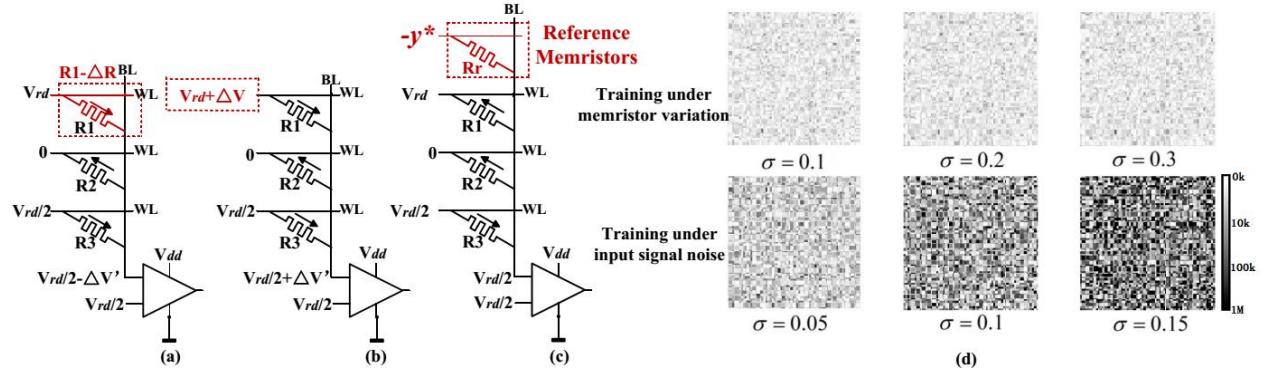


Figure 14. Training under memristor variation and input noise.

Our simulation shows very marginal degradation in the training robustness as the process variation increases. It is because the device variations are reflected in the difference between the current output and the target output during each iteration and compensated by close-loop training. Similarly, write pulse noise will cause memristance change variation in each iteration, which will also be compensated by close-loop training. However, input signal noise is generated on-the-fly and accumulated during the training process, leading to a large difference from the ideal trained result.

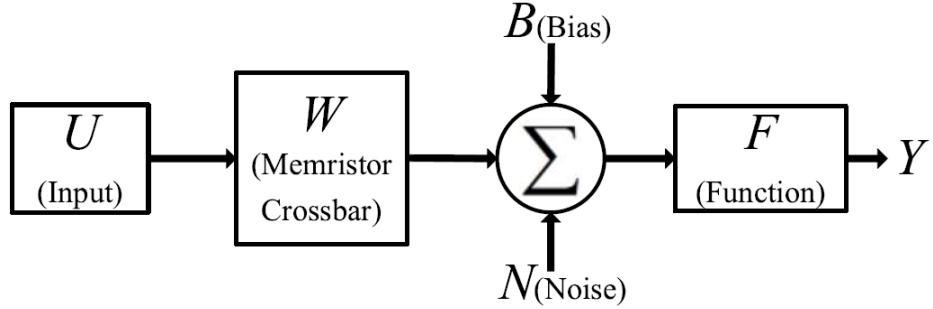


Figure 15. Training process with noise.

4.1.2 Noise Sensitivity of On-device Training

Figure 15 illustrates how this dynamic threshold training scheme works in system level. We assume F is the output activation function of the NCS, i.e., comparators, which translates the output of the crossbar to a digital value $\in \{1, -1\}$. The input signal noise N is added on the F before it is sent to the next iteration. Different from the conventional GDT, our method tries to minimize not only the 2-norm output distance $(y - y^*)^2$, but also the system's sensitivity

to the noise $\frac{\partial f(\sum u_i w_{ij})}{\partial N}$ as:

$$\underset{w}{\text{Min}} : J = \frac{J_1}{(y - y^*)^2} + \frac{J_2}{\frac{\partial f(\sum u_i w_{ij})}{\partial N}} \quad (14)$$

In the above cost function, J_1 and J_2 denote memristor crossbar output distance and the noise sensitivity, respectively. At the end of iteration t , the adjustment of the memristor crossbar in the next iteration $W_{(t+1)}$ can be derived from the current $W_{(t)}$ as:

$$\mathbf{W} = \mathbf{W} - \eta \frac{\partial J}{\partial \mathbf{W}} \quad (15)$$

or,

$$\mathbf{W} = \mathbf{W} - \eta \frac{\partial J}{\partial \mathbf{W}} - \eta \frac{\partial^2 f(\sum u_i w_{ij})}{\partial W \partial N} \quad (16)$$

The choice of training rate η is discussed in [30]. For the second term on the right of equation (16), we have:

$$\frac{\partial J_1}{\partial \mathbf{W}} = 2 \cdot (y - y^*) \cdot \frac{\partial y}{\partial \mathbf{W}} = 2 \cdot (y - y^*) \cdot U \quad (17)$$

Equation (17) means that the variations of \mathbf{W} (the process variation) is reflected by the output distance ($y - y^*$). J_2 is determined by the activation function f as:

$$\begin{aligned} \frac{\partial f(\sum u_i w_{ij})}{\partial N} &= \\ \lim_{\Delta n \rightarrow 0} f(\sum u_i w_{ij} + N) - f(\sum u_i w_{ij} - N) &\approx f'(\sum u_i w_{ij}) \end{aligned} \quad (18)$$

For the two popular activation functions in neuromorphic computing, i.e., sigmoid function and sgn function, equation (18) can be expressed as:

$$f'_{sigmoid}(\sum u_i w_{ij}) = e^{\sum u_i w_{ij}} / (1 + e^{\sum u_i w_{ij}})^2 \quad (19)$$

and

$$f'_{sgn}(\sum u_i w_{ij}) = \begin{cases} 1/\Delta N & \Delta N < \sum u_i w_{ij} < \Delta N \\ 0 & \text{else} \end{cases} \quad (20)$$

respectively. In both cases, the noise sensitivity decreases when $|\sum u_i w_{ij}|$ raises, as shown in Figure 16.

Figure 16.

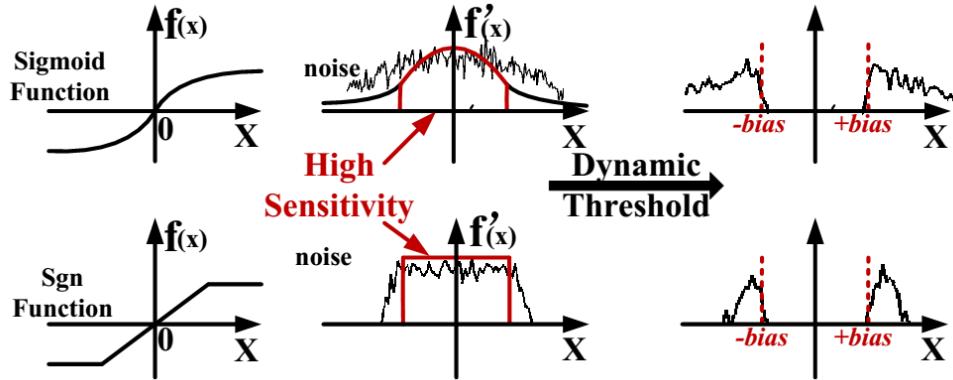


Figure 16. Noise elimination mechanism.

4.1.3 Noise-Eliminating Training Scheme

Based on our observation on equation (19) and (20), we proposed a noise-eliminating training scheme to minimize the noise accumulation during the on-device training. Redundant rows are added on top of the memristor array to generate an offset current \mathbf{B} that is opposite to the target output of the column y_i^* during on-device training, as shown in Figure 14 (c). It adds the bias to the calculated difference between the current output and the target output of the crossbar so that the $|\sum u_i w_{ij}|$ is shifted out of the sensitive region of $f(x)$ as:

$$x = \begin{cases} \sum w_{ij} w_{ij} u_i < \text{bias } y^* & \text{if } y^* = +1 \\ \sum w_{ij} u_i > \text{bias } y^* & \text{if } y^* = -1 \end{cases} \quad (21)$$

As shown in Figure 16, through applying *bias*, the residue of the noise in the sensitive region of the activation function is reduced and the accumulation of the noise during the training iterations is minimized. The selection of *bias* is important in our proposed scheme: A *bias* larger than necessary may make the training process bypass the convergence region, leading to the difficulty of convergence. If *bias* is too small, it may not efficiently suppress the noise. A detailed evaluation on the selection of *bias* will be given in Section 4.3.1.

We define bias *amplitude* a to measure the ability of the reference memristor to offset the crossbar output as:

$$a = \frac{R_{on} N_{ref}}{R_{ref} N_{col}} \quad (22)$$

Here R_{on} is the HRS of a memristor. R_{ref} is the average resistance of the reference memristors. N_{col} is number of memristors in a column. N_{ref} is the number of reference memristors in a column. During the on-device training, a training failure is defined as the unsuccessful convergence after the maximum n iterations of training. Here n is the threshold usually much more than the normal iteration number required for convergence. If a training failure happens, we will reset the reference memristors to reduce a and redo the training process until the training succeeds or $a=0$, which indicates the training is degraded to conventional training scheme.

4.2 DIGITAL INITIALIZATION

4.2.1 Basic Idea

In our noise-eliminating training scheme, the introduction of *bias* affects the convergence process of on-device training and may cause the potential convergence failure. In this section, we proposed a digital initialization step to the on-device training to reduce the training failure rate and training time.

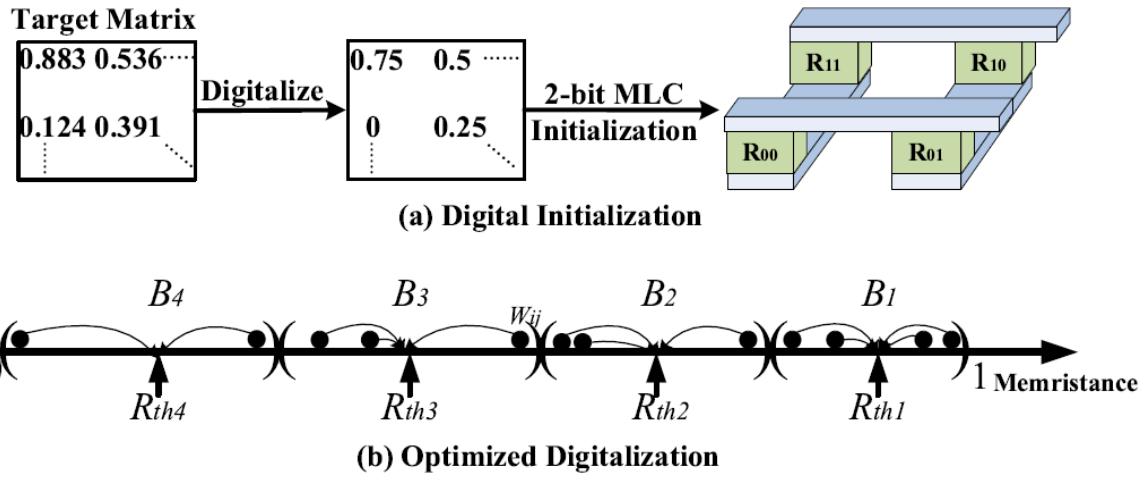


Figure 17. Digital initialization.

As shown in Figure 17, in the initialization step, the target W , which can be calculated beforehand, is quantized to its digital version \mathbf{W}_D where every element is represented by a multi-

level cell (MLC) data, e.g., 2-bit digit. \mathbf{W}_D is then written into the crossbar by the programming method introduced in section 2.3.2.1, regardless the device variations. Our digital-assisted training initialization step can improve the convergence speed of on-device training by setting the initial resistance of the memristors close to the target value. Different from the off-device training, the digital initialization does not require to program the memristor to the digitalized resistance level precisely and can tolerate the device variations. Note that the digitalization of \mathbf{W} relies on specific training algorithms as we will show next for our approach.

4.2.2 Digitalization of Weight Matrix

In the conventional MLC memory cell design, the distances between the two adjacent resistance states of the memristor must be the same to maximize the sense margin [69]. The threshold to differentiate the different MLC level is set to the cross point between the distributions of two adjacent resistance states. In on-device training, the convergence rate of the training process is conceptually determined by the distance between the target value and the initial value. Therefore, the partition method of MLC memory design does not necessarily give us the minimum distance in the digitalization of weight matrix \mathbf{W} .

We propose a heuristic method to determine the resistance states of the memristor corresponding to the different digitalized levels of \mathbf{W} : For an M -level digitalization, the elements of \mathbf{W} are equally classified into m baskets $B_i, i = 1 \dots m$ based on their values. We then find

the $R_{thi}, i = 1 \dots m$ for each basket to achieve the minimum $\sum |W_{ij} - R_{thi}|_{W_{ij} \in B_i}, i = 1 \dots m$,

$R_{th,i}$ is the optimal memristor resistance states for the i level of the digitalization. Here we used 1-norm resistance distance to measure the impact of the difference between W_{ij} and $W_{D,ij}$ on the overall convergence rate of the on-device training. For different on-device training algorithms, other methods, e.g., based on 2-norm distance or the maximum distance, may be also adopted. Considering the practical memristor programming resolution, we set $m=4$ here. Note that this method may cause smaller MLC sensing margin, however, we do not need to read out the value of each MLC. The initialization accuracy is enough to guarantee the training quality.

4.3 EXPERIMENTS AND RESULTS

4.3.1 Noise Elimination

Figure 18 illustrates the effectiveness of the noise-eliminating training method on improving the performance of memristor crossbar-based NCS. A hopfield network with 128 input neurons is built on a 128×128 crossbar with one-layer iterative structure to remember 16 patterns. We choose conventional delta rule (DR) training method for comparison. In our simulation, we set the *bias amplitude* a to 0.05. Monte-Carlo simulations are conducted under different process variations and input signal noise levels to measure the success rate when recognizing the image. As shown in Figure 18(a) and (b), even at the worst case of $\sigma_{variation} = 0.3$ or $\sigma_{variation} = 0.15$ at each comparison, our method still achieves the best performance.

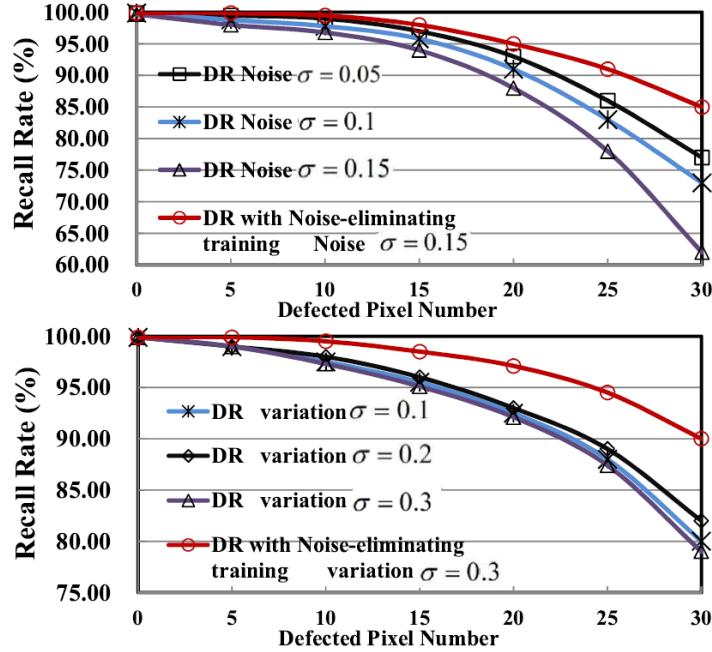


Figure 18. Effectiveness of noise-eliminating training.

4.3.2 Digital-Assisted Initialization

Figure 19 compares the training speed of the same on-device training simulated in Section 4.3.1. Y-axis is the Hamming distance between the output vectors of the crossbar and the target output vectors. X-axis is training iteration number. The size of training input vector set is 16 and the crossbar training ends when generated output matches the target patterns. Four combinations of process variations and input signal noise levels are simulated. To exclusively measure the effects of digital-assisted initialization, noise-eliminating training is not applied in the simulations.

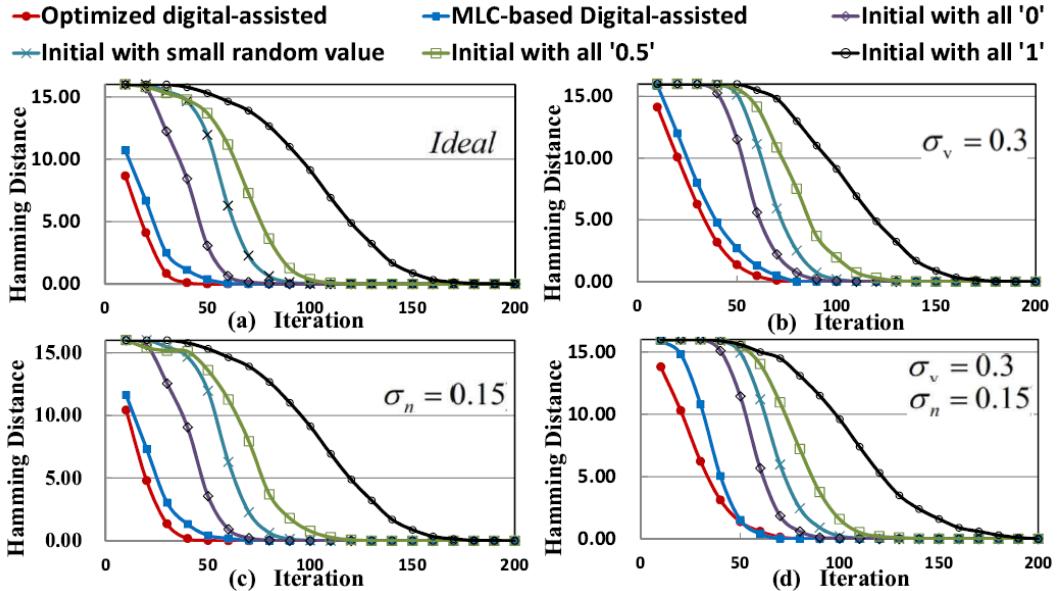


Figure 19. Comparison of convergence rate of different initialization.

Among all the simulated results, initializing the states of all memristors to ‘1’ (HRS) has the largest number of training iterations while initializing the states of all memristors to ‘0’ (LRS) has the smallest number of training iterations among all the simulations except the ones with the digital-assisted initialization. It indicates that the majority of the target memristor states are close to ‘0’.

“MLC-based digital-assisted” curve denotes the results of using the digitalization method of 2-bit MLC memory design in \mathbf{W} initialization while “Optimized digital-assisted” curve denotes the results of using the heuristic method proposed in Section 4.2. Both of them demonstrated much lower iteration number than the other training process without the digital-assisted initialization step. Our heuristic method offers the best result among all the training methods: when both process variation and input signal noise are considered, the training iteration number of “Optimized digital-assisted” is 23.3% less than that of “initialization with ‘0’”.

The introduction of process variation causes the deviation of the initial states of the memristors from the target states in the digital-assisted initialization step. It raises the Hamming distances of the first several iterations and increases the iteration numbers considerably, as shown in Figure 19 (b) and (d).

In general, the total training time of a conventional back propagation (BP) training method can be calculated by:

$$T_t = (T_p \cdot n^2 + T_c \cdot n) \cdot N_{iter}. \quad (23)$$

Here n is the input size of the crossbar. T_t is overall training time. T_p and T_c are the programming and comparison time consumed in each iteration. N_{iter} is the number of iterations.

When the digital-assisted initialization step is applied, the initialization time T_{init} is added to the total training time. Therefore, to achieve the positive benefit, the speed up introduced by the digital-assisted initialization step must be larger than the extra initialization time. Figure 20 shows that for a crossbar with the size of $n < 128$, digital-assisted initialization step does not give us any benefits on the training time reduction under the simulated conditions.

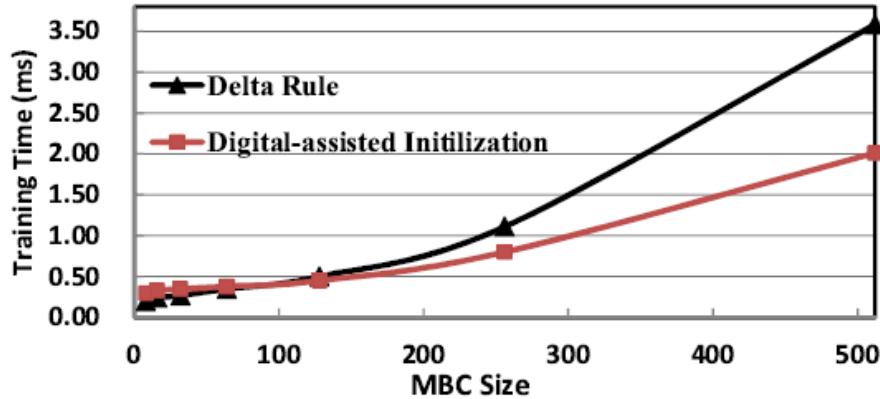


Figure 20. The impact of initialization on total training time.

4.3.3 Case study

To comprehensively evaluate effectiveness of all our proposed techniques, we implemented a three-layer feed forward neural based on a neuromorphic computing system with multiple 512×512 crossbars computing engines. BP training is used as comparison training

algorithm in this case. Other simulation parameters can be found at Table 1.

Table 1. Simulation setup

MBC	Single MBC Size	Input Voltage Noise		Memristor Variation
	512×512	$\sigma = 0.1$		$\sigma = 0.1$
Neuromorphic Engine	Neuron Number in Input Layer	Neuron Number in Hidden Layer		Neuron Number in Output Layer
	16384 ($32 \times$ MBC)	16384 ($32 \times$ MBC)		3
Peripheral Circuit	G	Vref (V)	Vdd/Gnd (V)	Vrd(V)
	50	0.6	1.2/0	0.1
	Amplifier Comparator			Registers
	Programming pulse voltage(V)	Programming pulse width(ns)	Number of Comparators/MBC	Number of Registers/MBC
	2.5/-2.5	50	512	512

Four sets of image patterns (e.g., face, animal, building and finger print) are adopted in the training neuromorphic computing systems. As shown in Figure 21, each pattern set has 8 images with a size of 128×128 pixels.

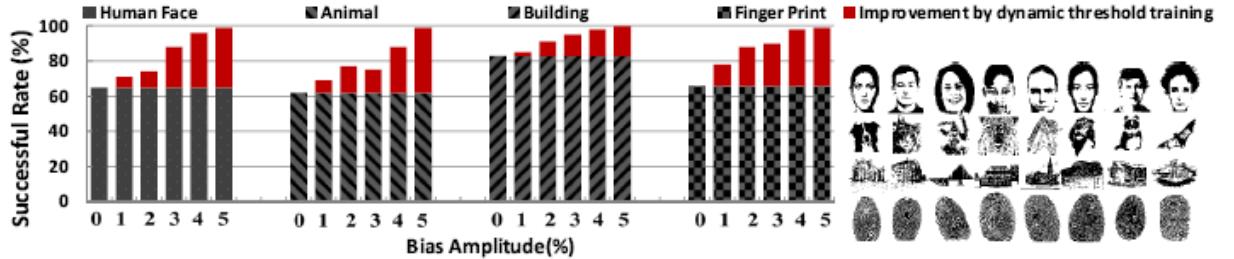


Figure 21. 3-layer network recall rate test of dynamic threshold training algorithm.

Figure 21 compares the recall success rates of the conventional back propagation (BP) training and the modified noise-eliminating method. Our method surpasses the conventional training method over all the simulation cases. Following the increase in the *bias amplitude*, the recall success rate improvement introduced by the noise-eliminating training method becomes more prominent.

Table 2. Training failure rate

Bias Amplitude "a"		0	0.01	0.02	0.03	0.04	0.05
Training Failure Rate "P _f " (%)	Human Face	0.61	1.9	4.2	7.3	15.4	31.5
	Animal	0.44	3.2	7.2	11.1	24.4	51.5
	Building	0.56	2.1	3.9	6.6	13.5	26.1
	Finger Print	0.39	2.7	4.5	9.1	19.4	41.7
Training Time "T _t " (ms)	Human Face	2.82	2.77	2.63	2.59	2.55	2.53
	Animal	2.76	2.57	2.56	2.40	2.28	2.25
	Building	2.66	2.5	2.48	2.34	2.31	2.28
	Finger Print	2.53	2.39	2.28	2.19	2.11	2.10

Table 2 shows the training failure rate and the training time (without digital-assisted initialization step) under the different *bias amplitude*. The increase in *bias amplitude* results in the reduction of the training time for each iteration while rapidly raises the training failure rate. As aforementioned in Section 3.3, Training failure will prolong the total training time since we will redo the training with a reduced *a*. The overall training time T_{train} will become:

$$T_{train} = T_{t(a=a_1)} + P_{f(a=a_1)} \cdot (T_{t(a=a_2)} + P_{f(a=a_1)} \cdot (T_{t(a=a_2)} + \dots) \quad (8)$$

where $T_{t(a=a_i)}$ and $P_{f(a=a_i)}$ are training time for each iteration and training failure rate for the training process with bias amplitude $a = ai$.

Figure 22 shows the overall training time comparison between conventional BP training, the modified noise-eliminating training with and without the digital-assisted initialization step starting with different *a*. Our techniques generally reduce the on-device training time by 12.6~14.1% for the same recall success rate, or improve the recall success rate by 18.7%~36.2%

for the same training time. Designer can pick the best combination based on the specific system requirement.

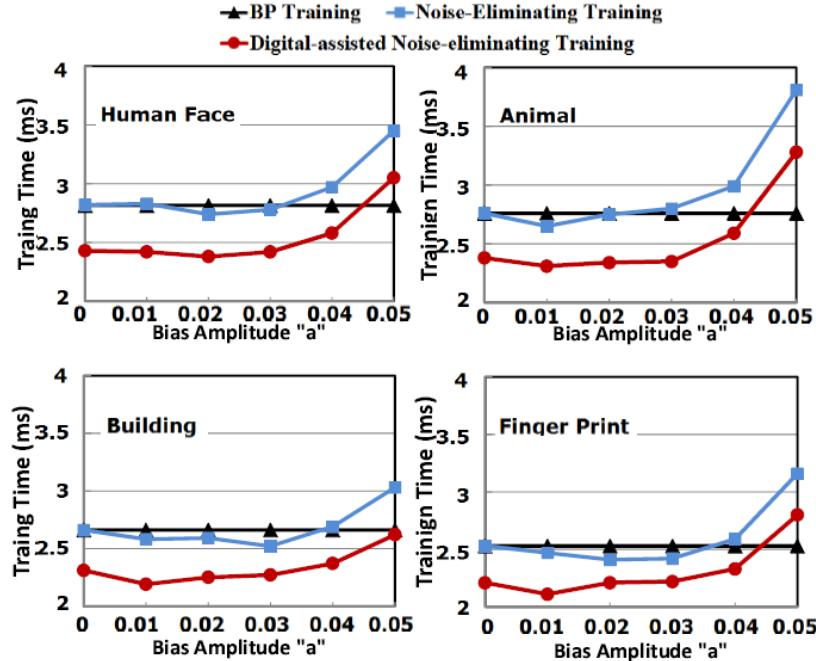


Figure 22. Comparisons of overall training time.

4.4 SECTION SUMMARY

In this section, we proposed a noise-eliminating training method and digital-assisted initialization step to improve the training process robustness and the performance of on-device training for memristor crossbar-based NCS. Experimental results show that our techniques can significantly improve the testing accuracy and training time of neuromorphic computing system by up to 18.7% ~ 36.2% and 12.6% ~ 14.1%, respectively, through suppressing the noise

accumulation in the training iterations and reducing mismatch between the initial weight matrix state and the target value.

5.0 SCALABILITY

5.1 IR-DROP LIMITS SINGLE CROSSBAR SIZE

5.1.1 Impact of IR-Drop on Memristor Crossbar

In a memristor crossbar, the voltage applied to the two terminals of a memristor is affected by the device location in the crossbar and the resistance states of all other memristors. In [57], the author explained that in the worst case, both sensing and programming of the crossbar will encounter severe reliability issues when the array size is beyond 64×64 . Although an NCS intrinsically can tolerate certain random errors in sensing process, IR-drop remains an issue in NCS training.

Figure 23 depicts the distribution of the actual voltage drop V' on each memristor in a 128×128 crossbar during the training process. Here $V_{bias} = 2.9V$. V'_{ij} is the voltage actually applied to the memristor between WL_i and BL_j. The largest IR-drop normally occurs at the far-end of the WL and BL (i.e., $V'_{(128,128)}$). The smallest/largest voltage degradation (IR-drop) occurs when all memristors are at their HRS/LRS. Figure 23 (b) shows that in the worst case, the largest IR-drop quickly increases to an unacceptable level as the crossbar size increases. It greatly decreases the programmability of the crossbar and degrades computation accuracy of the NCS. Degradation also occurs in recall process as shown in Figure 23 (c) and (d).

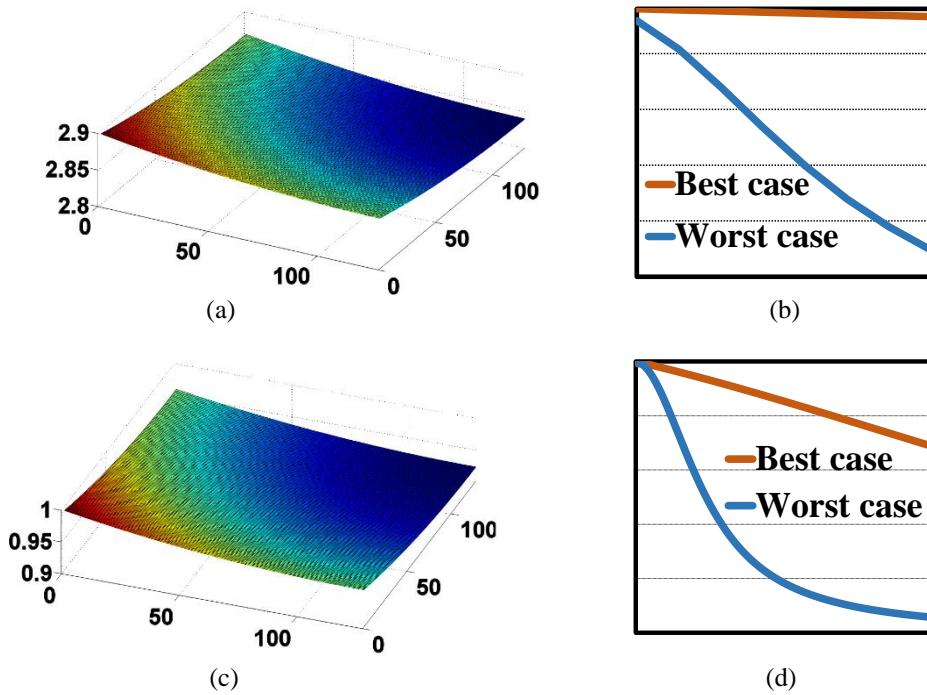


Figure 23. Voltage distribution with IR-drop.

5.1.2 Problem Formulation

5.1.2.1 Training

Normally the training of a crossbar starts with an initial state where all memristors are at their HRS. To program the initialized memristor crossbar (\mathbf{R}_{HRS}) to the target memristor resistance state \mathbf{R} that representing weight matrix \mathbf{W} , a training time matrix \mathbf{T} is generated based on the characterized relationship between the memristor resistance change and the programming time and voltage [48]:

$$\mathbf{R} = f(\mathbf{T}, \mathbf{V}, \mathbf{R}_{HRS}), \quad (24)$$

where \mathbf{V} is the ideal programming voltage ($V_{(i,j)}=V_{bias}$). After including the impact of IR-drop, the actual trained memristors resistance state is $\mathbf{R}'=f(\mathbf{T}, \mathbf{V}', \mathbf{R}_{HRS})$. Thus, if the \mathbf{V}' deviates from the ideal \mathbf{V} due to IR-drop, the actual trained crossbar \mathbf{R}' will be distinctive from \mathbf{R} . The difference between \mathbf{R} and \mathbf{R}' depends on the size of crossbar. As shown in Figure 2, when the programming voltage arriving at the memristor degrades from the ideal 2.9V to 2.7V (6.8% off). The programmed memristor resistance drifts from 900KΩ (point “A”) to 200KΩ (point “C”) at a programming duration of 0.4μs. More detailed experiments will be presented in Section 5.5.

5.1.2.2 Sensing

Normally during sensing process, the crossbar is read column by column, e.g., the WLs are connected to a certain input pattern and BLs are all grounded. As a result, the IR-drop induced voltage degradation demonstrates different patterns from that in training process. For example, when all WLs and BLs are respectively connected to 1 (1V) and 0 (GND), the ideal voltage distribution \mathbf{V} of a 128×128 crossbar should be an all-ones matrix and the ideal output will be:

$$\mathbf{y} = \mathbf{W} \circ \mathbf{V} \cdot \mathbf{x} = \mathbf{W} \cdot \mathbf{x}, \quad (25)$$

where “ \circ ” denotes the Hadamard product of two matrices and we assume the crossbar is ideally trained. However, as shown in Figure 23 (c), the actual voltage distribution \mathbf{V}^* deviates from \mathbf{V} and generates the actual output as:

$$\mathbf{y}^* = (\mathbf{W} \circ \mathbf{V}^*) \cdot \mathbf{x} = \mathbf{W}^* \cdot \mathbf{x}. \quad (11)$$

Here, we define \mathbf{W}^* as the distorted weight matrix producing the actual current output of the crossbar when IR-drop is taken into account. As crossbar is a pure resistance network, \mathbf{W}^* is a function of memristor resistance state and wire resistance R_{wire} :

$$\mathbf{W}^* = g(\mathbf{R}, R_{\text{wire}}). \quad (12)$$

Here \mathbf{R} is the target memristor resistance state. Figure 23 (d) shows that \mathbf{y}^* is directly determined by \mathbf{V}^* .

5.2 SYSTEM REDUCTION

The impact of IR-drop is heavily determined by the size of the crossbar. Hence, if we can reduce the scale of the involved computation on the crossbar, the required size of the crossbar will decrease and the computation reliability of the NCS will improve. In [46], partitioning of large-scale crossbar systems was proposed to address the gap between the data size of applications and the size limit of crossbar implementation. However, partitioning often incurs additional circuit cost, e.g., analog signal transmission, which is associated with large area and energy overheads. Besides, the precision of the signal inevitably degrades during the analog data transmission between each crossbar blocks. Reducing the crossbar size, hence, offers a transparent enhancement to crossbar partitioning by alleviating the precision requirements on signal transmission and processing in the crossbar.

5.2.1 Weight Matrix Approximation

The first step of our proposed crossbar system reduction scheme is to approximate the weight matrix \mathbf{W} ($n \times m$) in equation (1). In general, for any given weight matrix \mathbf{W} , we can leverage singular value decomposition (SVD) method to approximate \mathbf{W} as [66]:

$$\mathbf{W} = \mathbf{U} \sum \mathbf{V} \approx \mathbf{W}_{appx} = \sum_{i=1}^r \delta_i \cdot \mathbf{u}_i \cdot \mathbf{v}_i, \quad (26)$$

where \mathbf{U} and \mathbf{V} are unitary matrices, Σ is an rectangular diagonal matrix with singular values of \mathbf{W} . δ_i ($i=1, \dots, r$) are the first r (i.e., the rank of \mathbf{W}_{appx}) singular values of \mathbf{W} . \mathbf{u}_i and \mathbf{v}_i are the approximate left and right singular vectors of \mathbf{W} [66], respectively. The sequence of δ_i indicates the weights of each item of $\mathbf{u}_i \cdot \mathbf{v}_i$. By collecting a few multiplication product of \mathbf{u}_i (an $n \times 1$ vector) and \mathbf{v}_i (a $1 \times m$ vector), we can obtain a very good approximation of \mathbf{W} . The difference between \mathbf{W} and \mathbf{W}_{appx} , that is $\Delta\mathbf{W} = \|\mathbf{W} - \mathbf{W}_{appx}\|$, is decided by the coverage of $\sum_{i=1}^r \delta_i$ on the overall summed $\sum_{i=1}^m \delta_i$. The difference, hence, $\Delta\mathbf{W}$ can be controlled by the value of r .

In general, a larger rank r leads to a better approximation of \mathbf{W} but increases crossbar size and training time cost. However, increasing r does not necessarily result in a more robust crossbar hardware implementation due to the following reasons: First, r is limited by an upper bound, say, the rank of \mathbf{W} . Increasing r beyond this upper bound is meaningless; Second, r solely defines the size of one dimension of the reduced crossbar (say, $n \times r$, which will be shown in the next section). Increasing r will directly aggravate the impact of IR-drop.

For the above reasons, a threshold “ ε ” of singular value coverage is heuristically predefined for r selection during the approximation of \mathbf{W} . Here $\varepsilon \in [0,1]$. r is then selected as:

$$\min: r, \text{s.t. } 1 - \frac{\sum_{i=1}^r \delta_i}{\sum_{i=1}^m \delta_i} < \varepsilon. \quad (27)$$

The efficacy of this select strategy will be shown in Section 5.5.2.

5.2.2 One-dimensional (1-D) Reduction

Based on the approximation result, we are able to transform the weight connection function in equation (1) as:

$$\begin{aligned} \mathbf{W} \cdot \mathbf{x} &\approx (\sum_{i=1}^r \delta_i \cdot \mathbf{u}_i \cdot \mathbf{v}_i) \cdot \mathbf{x} \\ &= (\delta_1 \cdot \mathbf{u}_1) \cdot (\mathbf{v}_1 \cdot \mathbf{x}) + (\delta_2 \cdot \mathbf{u}_2) \cdot (\mathbf{v}_2 \cdot \mathbf{x}) \dots (\delta_r \cdot \mathbf{u}_r) \cdot (\mathbf{v}_r \cdot \mathbf{x}) \\ &= [\delta_1 \cdot \mathbf{u}_1 \dots \delta_r \cdot \mathbf{u}_r] \cdot \begin{bmatrix} \mathbf{v}_1 \cdot \mathbf{x} \\ \vdots \\ \mathbf{v}_r \cdot \mathbf{x} \end{bmatrix} = [\delta_1 \cdot \mathbf{u}_1 \dots \delta_r \cdot \mathbf{u}_r] \cdot \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_r \end{bmatrix} \cdot \mathbf{x} \quad (28) \\ &= \mathbf{W}_{left} \cdot \mathbf{W}_{right} \cdot \mathbf{x}, \end{aligned}$$

where

$$\mathbf{W}_{left} = [\delta_1 \cdot \mathbf{u}_1 \dots \delta_r \cdot \mathbf{u}_r], \mathbf{W}_{right} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_r \end{bmatrix}. \quad (29)$$

Here \mathbf{W} was originally represented on an $n \times m$ crossbar and $m \times 1$ vector \mathbf{x} is represented by the input voltage vector of the crossbar.

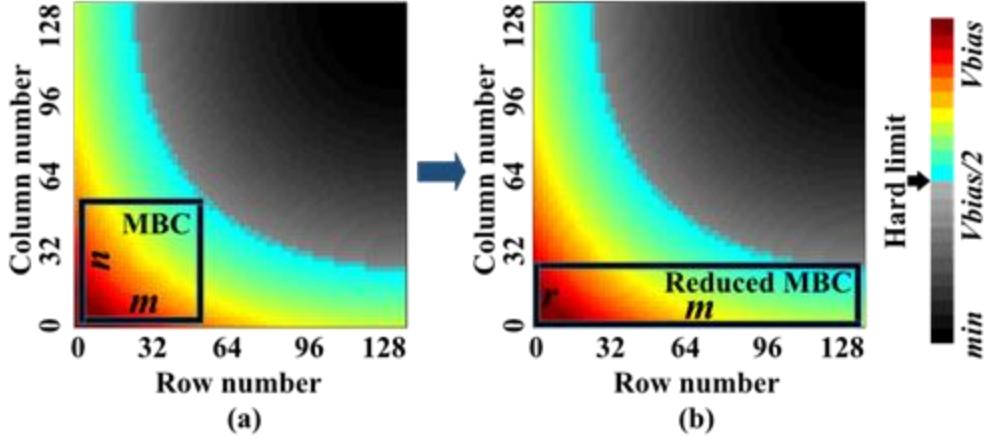


Figure 24. System reduction improves reliability.

Equation (28) and (29) show that the connection function can be transformed to a new two-stage system that consists of a $n \times r$ weight matrix \mathbf{W}_{left} and a $r \times m$ weight matrix \mathbf{W}_{right} . Note that $r \ll n$ or m . We named this method as one-dimensional (1-D) reduction, which has several significant advantages: First, after the original $n \times m$ memristor array is divided into two smaller arrays of $n \times r$ and $r \times m$, respectively, the programming time of the NCS is reduced from $O(n \times m)$ to $O(n+m)$; Second, 1-D reduction significantly improves the programming robustness of an single crossbar, which can be described below:

Figure 24 depicts the programming voltage drop distribution on a 128×128 crossbar, similar to Figure 24 (a). The only difference is that in Figure 24, all memristors are at LRS, demonstrating the worst-case impact of IR-drop on crossbar programming. As aforementioned in Section 5.1, during the programming of the crossbar, the voltage reaching the target memristor may degrade from the original programming voltage when IR-drop is considered. And the degradation level relies on the location of the target memristor. In Figure 24, we highlight (colored) the memristor locations with a voltage drop higher than $V_{bias}/2$ when the voltages of

the WL and BL connecting the memristor are set to $+V_{bias}$ and GND , respectively. We name the boundary of the highlighted area as the “hard-limit” of a single crossbar scale. Any memristors outside the “hard-limit” will not be effectively programmed because they are practically “half-selected” (see Section 2.3). Increasing the programming voltage to raise the voltage applied on the memristors outside the “hard-limit”, however, will affect the memristors that should be “half-selected”. Hence, the scale of the “hard-limit” serves as a good measurement of crossbar programming robustness. As shown in Figure 24 (a), the largest crossbar size within the “hard-limit” is only 48×48 by assuming the sizes of the two dimensions of the crossbar are the same, i.e., $n = m$. The maximum size of the data can be processed is only 48. If we can reduce the size of one dimension down to a smaller value, say, $r = 22$, then the size of the another dimension can be extended to 128, as shown in Figure 24 (b). Such a crossbar is sufficient to process the data with a size of 128 by leveraging our proposed 1-D reduction method, as long as the rank of \mathbf{W}_{appx} is not higher than 22.

5.2.3 Two-dimensional (2-D) Reduction

1-D reduction can downscale the size of the needed crossbar from $n \times m$ to $n \times r$ and $r \times m$, resulting in significant saving on the hardware design cost and better robustness. In some pattern classification tasks, we may further reduce the crossbar size in both dimensions. For example, when classifying a noisy input pattern \mathbf{x}_i (e.g., an $n \times 1$ vector), weighted network connections (a $n \times n$ matrix) are needed to associate a noisy input pattern to as one of the standard pattern \mathbf{a}_q ($q=1, 2, \dots, r$). Our proposed two-dimensional (2-D) reduction can further reduce the scale of the

computing system by transforming the concerned neuromorphic algorithm to a distance comparison based classification as follows:

Without loss of generality, the similarity between the output vector $\mathbf{W} \cdot \mathbf{x}_i$ and the standard pattern vector \mathbf{a}_q ($q=1,2,\dots,r$) can be quantitatively measured by:

$$P_{iq} = (\mathbf{W} \cdot \mathbf{x}_i)' \cdot \mathbf{a}_q = (\mathbf{a}'_1 \cdot \mathbf{x}_i) \cdot (\mathbf{a}'_1 \cdot \mathbf{a}_q) + \dots + (\mathbf{a}'_r \cdot \mathbf{x}_i) \cdot (\mathbf{a}'_r \cdot \mathbf{a}_q). \quad (30)$$

Similar to equation (28), we can use $\mathbf{a}'_q \cdot \mathbf{x}_i$ ($q = 1, 2 \dots r$) to form a new input vector

$\tilde{\mathbf{x}}_i$ and calculate the similarity between $\tilde{\mathbf{x}}_i$ and other patterns as:

$$P_i = \begin{bmatrix} P_{i1} \\ \vdots \\ P_{ir} \end{bmatrix} = \begin{bmatrix} \mathbf{a}'_1 \cdot \mathbf{a}_1 & \dots & \mathbf{a}'_r \cdot \mathbf{a}_1 \\ \vdots & \ddots & \vdots \\ \mathbf{a}'_1 \cdot \mathbf{a}_r & \dots & \mathbf{a}'_r \cdot \mathbf{a}_r \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a}'_1 \cdot \mathbf{x}_i \\ \vdots \\ \mathbf{a}'_r \cdot \mathbf{x}_i \end{bmatrix} = \widehat{\mathbf{W}} \cdot \begin{bmatrix} \mathbf{a}'_1 \cdot \mathbf{x}_i \\ \vdots \\ \mathbf{a}'_r \cdot \mathbf{x}_i \end{bmatrix}. \quad (31)$$

where $\widehat{\mathbf{W}}$ is the new weight matrix with a dimension of $r \times r$.

Equation (31) implies that after the proposed 2-D reduction, the size of the needed crossbar is no longer determined by the large dimension size of data pattern (n) but the number of the patterns needs to be trained (r). *This new property is of particularly importance to applications that process the data with large dimensions but only limited number of patterns to be concerned, e.g., identifying objects on high resolution image or video.*

5.2.4 Implementation Example

The proposed 1-D reduction scheme is applicable to any network models that contain the operation described in equation (1) while the 2-D reduction scheme can fit in some applications like Auto-Associate Memory (AAM) well. Here we use Hopfield-network as an example to

illustrate the basic concept of hardware implementation of the two proposed system reduction schemes.

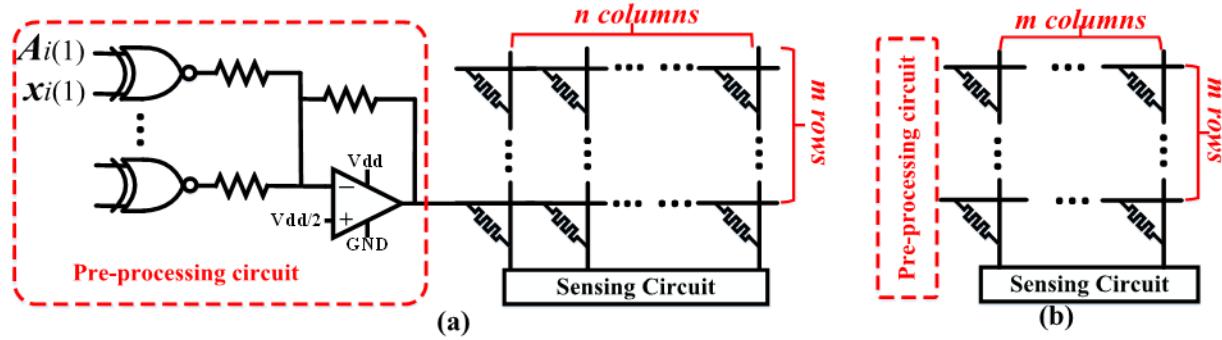


Figure 25. Conceptual schematics of (a) 1-D reduction (b) 2-D reduction.

Conventional Hopfield-network uses recurrent data process architecture to implement associative memory by training the connecting synapse weights based on stored standard patterns [26]. Each of the neurons has an activation function $f(x_{input})$ as:

$$x_{output} = \begin{cases} +1 & \text{if } x_{input} \geq \theta \\ -1 & \text{if otherwise} \end{cases} \quad (32)$$

where θ is the activation threshold, which determines whether this neuron fires an excitation or not. The input of each neuron is the summation of the activations from all the neurons of the synapse network during last iteration. In general, data processing through the synapse network can be expressed as:

$$x_{i+1} = f(W \cdot x_i), \quad (33)$$

where \mathbf{x}_i and \mathbf{x}_{i+1} are $n \times 1$ vectors that denote the states of n neurons in the current time cycle i and the next time cycle $i+1$, respectively. Weight matrix \mathbf{W} is trained with Hebbian rule as shown in (30).

Figure 25 shows the conceptual schematic of our proposed system reduction schemes, including both 1-D and 2-D designs. For the purpose of demonstration, here we assume that the inputs of the NCS, i.e., \mathbf{x}_i , are all binary information (0 or 1). Both 1-D and 2-D reduction schemes require the inputs to be preprocessed by multiplying with the concerned patterns. In normal implementation of Hopfield network, the outputs of the crossbar are directly sent to comparators which conduct “*sign*” function. In our reduction schemes, a slightly more complex post-processing is performed at the outputs, which can be implemented with a traditional analog selecting circuit. The analysis of system robustness and implementation area tradeoff will be presented in Section 5.5.

Compared to the 1-D reduced weight matrix $\widetilde{\mathbf{W}}$, the 2-D reduced weight matrix $\widehat{\mathbf{W}}$ is more sensitive to memristor device variations as the variability of one memristor has relatively higher impact on the computation accuracy due to significantly reduced number of the memristors participating in the computation. More details on the design tradeoffs between the two reduction schemes will be discussed in Section 5.5, together with the discussion on the design cost of the associated extra peripheral circuitry.

5.3 IR-DROP COMPENSATION

In addition to reducing the dimension sizes of the crossbar, we can also actively compensate the impact of IR-drop to further improve the computation reliability of the NCS. In this section, we propose an adaptive compensation method that can compensate the impact of IR-drop in both training and sensing processes.

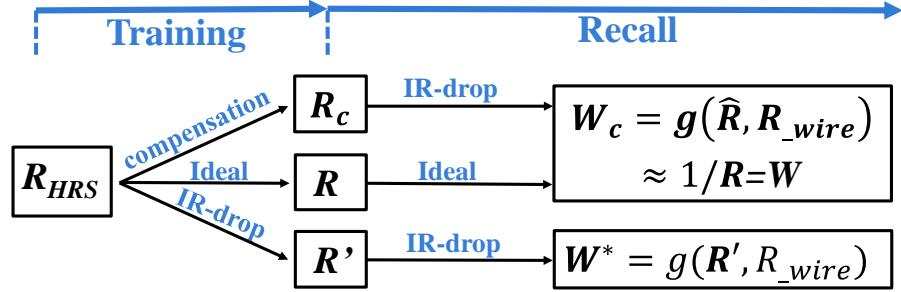


Figure 26. Compensation for both training and sensing process.

5.3.1 Sensing Compensation

Based on equation 12, the weight matrix represented by a crossbar with resistance state \mathbf{R} is not $\mathbf{W} = \mathbf{I}/\mathbf{R}$ but $\mathbf{W}^* = g(\mathbf{R}, R_{wire})$ when IR-drop is considered. As summarized in Figure 26, the IR-drop compensation can be performed by searching the new memristor crossbar resistance state \mathbf{R}_c that generates a weight matrix \mathbf{W}_c closest to the ideal target \mathbf{W} as:

$$\min_{\mathbf{R}_c} \overbrace{\|\mathbf{W} - \mathbf{W}_c\|_F}^{F(\mathbf{R}_c)}^2 = \sum_{i=1}^n \sum_{j=1}^m (\mathbf{W}_{(i,j)} - \mathbf{W}_{c(i,j)})^2. \quad (34)$$

Here, we define a cost function $F(\mathbf{R}_c)$ as the square of F-norm distance. \mathbf{W} is a $n \times m$

matrix. This optimization problem can be solved by the gradient search method with \mathbf{R}_c starting from $\mathbf{R}_{c_k} = \mathbf{R}$ as:

$$\begin{aligned} \mathbf{R}_{c_{k+1}} &= \\ \mathbf{R}_{c_k} - \gamma \nabla F(\mathbf{R}_{c_k}) &= \mathbf{R}_{c_k} + \gamma \cdot \sum_{i=1}^n \sum_{j=1}^m (2 \cdot (\mathbf{W}_{(i,j)} - \mathbf{W}_{c(i,j)}) \cdot \frac{\partial \mathbf{W}_{c(i,j)}}{\partial \mathbf{R}_{c_k}}), \end{aligned} \quad (35)$$

where γ is the step length. The gradient direction relies on the relation between \mathbf{W}_c and \mathbf{R}_c , i.e.,

$\mathbf{W}_c = g(\mathbf{R}_c, R_{\text{wire}})$. Here g is a function that can be explicitly measured as follows: when we apply 1V on i -th WL of a crossbar with resistance state of \mathbf{R}_c and wire resistance of R_{wire} and ground all other WLs and BLs, the magnitudes of the output current from BLs are equal to the elements in the i -th row of \mathbf{W}_c .

In general, the currents from every BL can be calculated by Modified Nodal Analysis as [71]:

$$\mathbf{Y}(\mathbf{R}_c, R_{\text{wire}}) \cdot \begin{bmatrix} \mathbf{v} \\ \mathbf{k} \end{bmatrix} = \begin{bmatrix} \mathbf{i} \\ \mathbf{e} \end{bmatrix}. \quad (36)$$

$$\mathbf{Y}_{(i,j)} = \sum_{i=1}^n \sum_{j=1}^m (a_{(i,j)} / \mathbf{R}_{c(i,j)}) + b / R_{\text{wire}}. \quad (37)$$

Here \mathbf{Y} denotes a conductance matrix that is a polynomial function of \mathbf{R}_c and R_{wire} . \mathbf{v} is the vector of total $2 \times n \times m$ node voltages. \mathbf{k} is the vector of $n+m$ WL/BL currents. \mathbf{i} is the vector of current sources at each node, most of which are zeros except for the elements corresponding to WL/BL ports. \mathbf{e} is the vector of $n+m$ voltage sources ($e_{(i,1)}=1V$, other=0V). Then we have,

$$\mathbf{W}_{c(i,j)} = \mathbf{k}_{(n+j,1)} \quad (j = 1 \dots m). \quad (38)$$

For the last term in equation (35), we have:

$$\frac{\partial \mathbf{W}_{c(i,j)}}{\partial R_c} = \frac{\partial \mathbf{W}_{c(i,j)}}{\partial Y(R_c, R_{\text{wire}})} \cdot \frac{\partial Y(R_c, R_{\text{wire}})}{\partial R_c} = \frac{\partial \mathbf{k}_{(n+j,1)}}{\partial Y(R_c, R_{\text{wire}})} \cdot \frac{\partial Y(R_c, R_{\text{wire}})}{\partial R_c}, \quad (39)$$

where $\partial Y(R_c, R_{\text{wire}})/\partial R_c$ can be directly calculated based on equation (37). $\partial \mathbf{k}_{(n+j,1)}/\partial Y(R_c, R_{\text{wire}})$ is the sensitivity of current \mathbf{k} to the conductance parameters \mathbf{Y} in equation (36). This sensitivity can be solved by Adjoint Sensitivity Analysis (ASA) . Figure 27 demonstrated an example about how the impact of IR-drop in a 64×64 memristor crossbar is compensated. Simulation results show that the difference between \mathbf{W}_c and \mathbf{W} ($\|\mathbf{W} - \mathbf{W}_c\|_F^2$, as Y-axis) can be reduced down to below 1% only within 6 update steps described in equation (35).

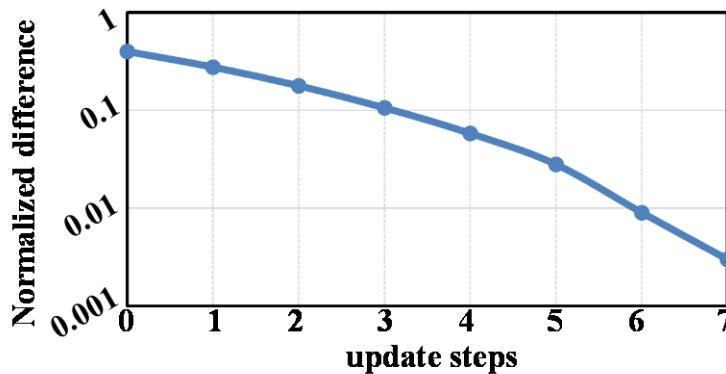


Figure 27. Sensitivity analysis based compensation.

5.3.2 Training Compensation

The objective of IR-drop compensation during crossbar programming is to minimize the difference between the trained resistance state \mathbf{R}' of the memristors and the ideal resistance state \mathbf{R} that represents target weight matrix \mathbf{W} . According to [57], IR-drop leads to minimum voltage degradation when all memristors are set to HRS. Thus, before training starts, all the memristors in the crossbar should be initialized to HRS (\mathbf{W}_{HRS}) to minimize the impact of IR-drops.

We define the ideal training time matrix \mathbf{T} as the required programming pulse widths on the memristors and \mathbf{V} as the ideal training voltage distribution applied on the memristors without considering IR-drop. \mathbf{R} is the function of \mathbf{T}, \mathbf{V} , and \mathbf{R}_{HRS} or $\mathbf{R} = f(\mathbf{T}, \mathbf{V}, \mathbf{R}_{HRS})$. Here T_{ij} is the programming pulse width applied on the memristor connected by WL_i and BL_j. f is the memristor switching function that can be derived from Figure 2. However, when IR-drop is considered, the training voltage distribution matrix is distorted to \mathbf{V}' . To compensate the voltage degradation in \mathbf{V}' , we can first calculate \mathbf{V}' before programming each memristor and then derive a new training time matrix \mathbf{T}' in order to obtain a trained crossbar \mathbf{R}' close to \mathbf{R} . For example, when programming voltage reduces from 2.9V to 2.7V (see Figure 2), the required programming time needs to be extended from 500ns to 3s to program the memristor to the same resistant state.

5.4 MODEL COMPRESSION

There are two ways to improve scalability of a system: increasing the capability of the hardware and compressing the model that needs to be mapped onto hardware. In this section, we will use model compression technology to increase NCS scalability from a different angle.

Usually, the best performing machine learning models are ensembles of multiple base models. Then, a more accurate and robust prediction can be produced based on voting results of base models. However, the large ensemble model requires significant amount of storage and computing capability to make a fast prediction. This motivated [77] to propose the technique of “model compression”, which gave a promising answer to the question “how to run a large ensemble model faster with less storage cost”. Their answer is a compression process that trains a smaller and faster model by approximating the function learnt by the well trained large ensemble models.

As shown in Figure 28, the compact single model can be trained with the original training samples in a conventional procedure. Once the large ensemble model is trained, there will be an alternative set of training data we can use.

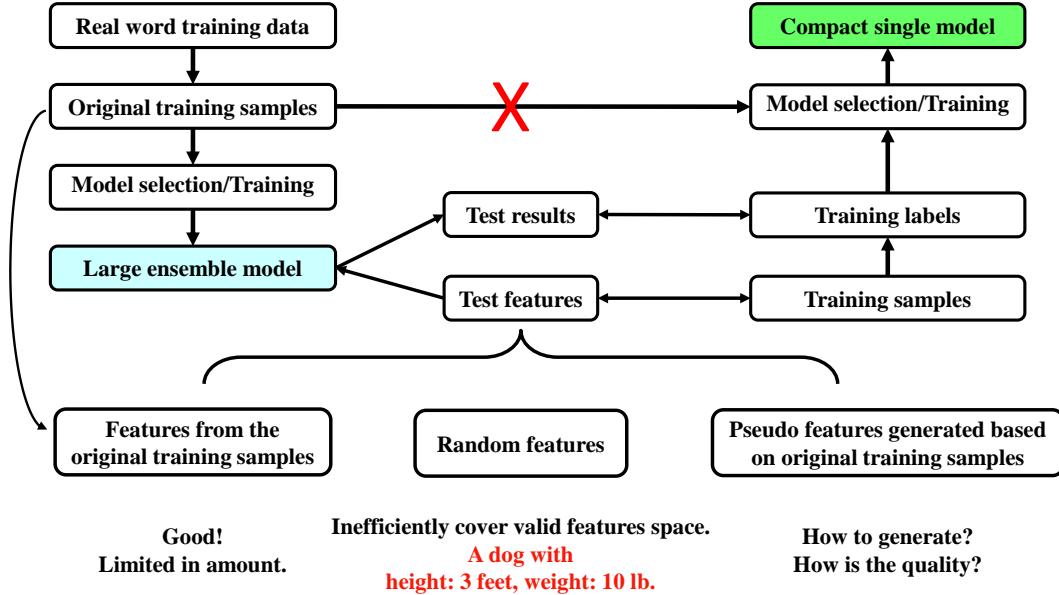


Figure 28. Model compression.

Unlike the true function underlying the original real world training data that is unknown, the function learned by a high performing model is available and can be used to test on any input feature. To prepare a new set of training data, which we name as “pseudo data”, we use the large ensemble model to test on and label the “Test features” in Figure 28. There are three different approaches to generate Test features:

- 1) Use the features from the original training samples: This is the most conservative way to obtain features. However, the original training set is limited in size and the model compress can possibly consume more data than the conventional training.
- 2) Randomly generate features: Randomly generated features are not limited in amount. However, this method may be very inefficient as it may generate a lot of “invalid”

features. For example, a sample of dog can be generated to have height of 3 feet with weight of 10 lb., which is very unlikely to happen in reality.

- 3) Pseudo features generated base on original training samples: This method induce random changes to the original features to extend the volume of the original samples. We will show examples and experiment results in section 5.5.6.

Then, the compact single model can be trained based on the pseudo data and this allows a slow, complex model such as a massive ensemble to be compressed into a fast, compact model such as a neural net with little loss in performance. Model compression can be directly used to reduce the model complexity or size of memristor crossbar based NCS. More experiment results can be found in section 5.5.6.

5.5 EXPERIMENTAL RESULTS

In this section, we will evaluate the effectiveness of the proposed schemes through a set of experiments: Section 5.5.1 shows the training quality improvement via IR-drop compensation; Section 5.5.2 defines reading accuracy and discusses the selection of rank r in system reduction; Section 5.5.3, 5.5.4 and 5.5.5 evaluate implementation area, performance and robustness of both system reduction methods, respectively. The trade-off between two methods will be particularly discussed in Section 5.5.5; Section 5.5.6 gives a study on the model compression technique. Table 3 summarizes the parameters of the memristors and crossbar designs used in our simulations.

Table 3. Experiment parameters.

Parameters		Values
R_{wire} for $2F \times 2F$		2.5Ω
Amplifier area		5340 F^2 [19]
XOR gate area		280 F^2
Memristor	High resistance state	$1 \text{ M}\Omega$
	Low resistance state	$10 \text{ K}\Omega$
	Cell area	4 F^2
Programming pulse amplitude(V_{bias})		2.9V
Reading voltage		1V

5.5.1 Training Quality

In a crossbar, the voltage applied to the two terminals of a memristor is affected by the device's location in the crossbar Figure 29. Trained resistance discrepancy. Figure 29 shows the simulation results on the memristor resistance discrepancy between the target crossbar and the actual trained crossbar under the impacts of IR-drop and process variations. Similar to the training voltage degradation pattern shown in Figure 23, the largest memristor resistance discrepancy occurs at the far end of the crossbar. Here we assume that the programmed memristor resistance follows the log-normal distribution as $r = r_0 \cdot \exp(\theta)$ [39]. r_0 stands for the mean of the programmed memristor resistance and $\theta \sim N(0, \sigma)$ is a random variable that follows Gaussian distribution.

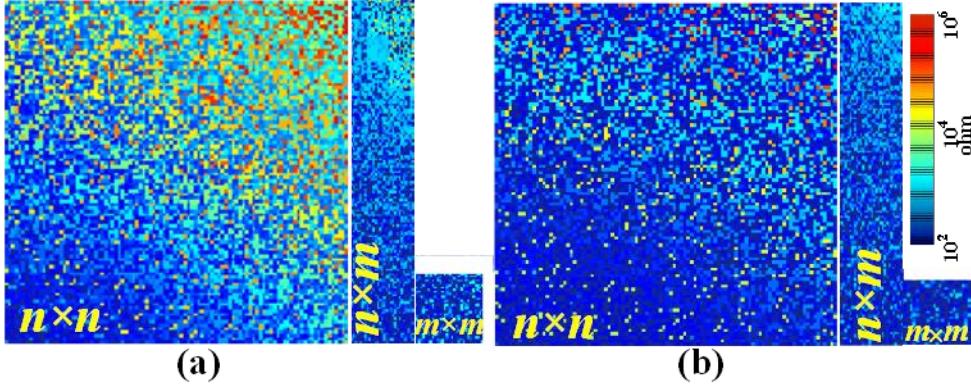


Figure 29. Trained resistance discrepancy.

We use the example from Section 5.2.4 to illustrate the design of NCS. The crossbar scale can be reduced from 128×128 (original $n \times n$) down to 128×19 ($n \times r$) and 19×19 ($r \times r$) by applying 1-D and 2-D reduction schemes, respectively. Here r is selected as 15% n , which is the maximum pattern numbers that can be stored in a 128×128 Hopfield network in theory. As shown in Figure 29 (a), the memristor resistance discrepancy significantly reduces when crossbar size decreases, implying a better training quality.

To further enhance NCS training quality, we introduce the IR-drop compensation technique given in Section 5.3.1 into training process. Figure 29 (b) shows that the compensation technique effectively minimizes the memristor resistance discrepancy. As we shall show in Section 5.5.5, the training quality enhancement can substantially improve recall successful rate (testing accuracy) of the NCS.

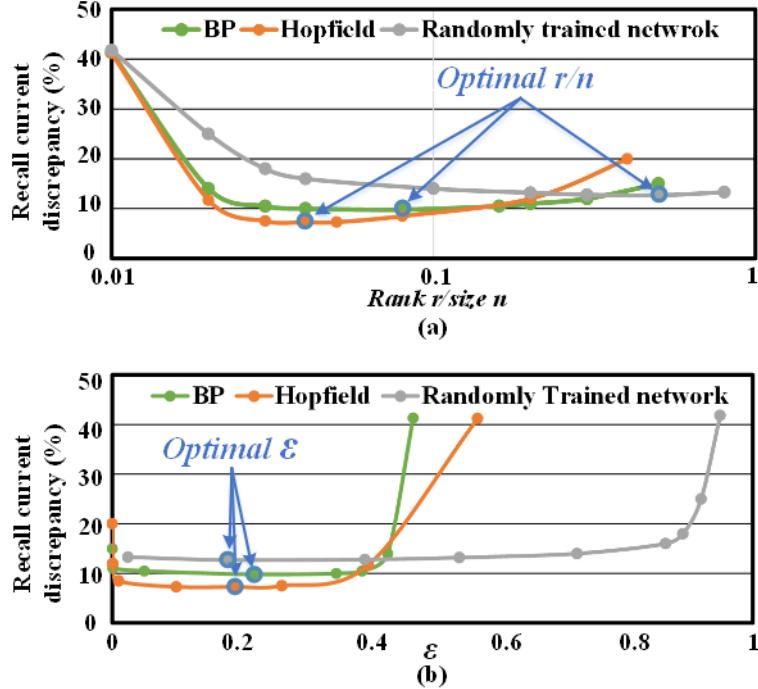


Figure 30. Recall discrepancy (a) respect to r/n , (b) respect to ε .

5.5.2 Reading Accuracy and Selection of r

In this experiment, reading of a crossbar is defined as the case that all WLs of the crossbar are connected to 1V while all BLs are grounded. In such a case, the ideal output current from the BLs should equal $\mathbf{W} \cdot \mathbf{x}$ (\mathbf{x} is all one vector). However, due to IR-drop, the actual output

current \mathbf{I} will show deviation from $\mathbf{W} \cdot \mathbf{x}$, which can be described as reading accuracy issue. In

this experiment, we evaluate the reading accuracy of the proposed reduction schemes under different conditions. We will also discuss the rank selection (r) of the 1-D reduced weight matrix based on the read accuracy. To achieve the maximum representation, the benchmarks adopted in the experiment include Hopfield network, BP training based weight connection and random

weight matrix, all of which have a size of 100×100 . We program the crossbar to target weight matrix \mathbf{W} for different benchmarks under impact of same memristor variation as section 5.5.1.

We first scan r from 1 to 100 and see how the value of r/n ($n=100$) affects reading discrepancy, i.e., $|\mathbf{I} - \mathbf{W} \cdot \mathbf{x}| / |\mathbf{W} \cdot \mathbf{x}|$. Based on the experiment results shown in Figure 30 (a), the optimal r varies significantly in different benchmarks. For example, Hopfield network reaches the lowest read discrepancy when $r/n=0.08\sim0.19$, while random weight matrix reaches it when $r/n=0.3\sim0.8$. So finding a generic proper range of r/n for all benchmarks becomes impossible. The main reason is because each benchmark has different distributions of SVD singular value. The singular value sequence of the Hopfield network used in our experiment, for example, is 53.7, 4.6, 3.8, 3.2.... while that of the random weight matrix is 5.03, 2.89, 2.77, 2.62..... Due to the highly skewed distribution of the singular values, a small r (low rank) is sufficient for the Hopfield network. The random weight matrix, however, needs a large r because of the relatively similar singular values.

We note that the threshold ε introduced in equation (27) serves a generally good guidance for the rank selection. Figure 30 (b) shows the read accuracy degradation followed by the increase of ε . The optimal values of ε for the three benchmarks all locate within the range of [0.1, 0.3]. Note that $\varepsilon = 0$ means r equals the rank of the original weight matrix \mathbf{W} . Continue increasing r beyond the rank of \mathbf{W} will not improve the read accuracy of the crossbar but introducing extra IR-drop and noise. In the following experiments, we heuristically choose r by setting ε to 0.2.

5.5.3 Training Performance

Neither system reduction nor IR-drop compensation will affect the sensing time of the NCS. However, training time can be affected by both techniques. Here we still use the example from Section 5.2.4. Figure 31 compares the training times of the corresponding NCS designs with and without system reduction techniques and the training compensation time overheads. As memristors are programmed one by one in crossbar, system reduction naturally shortens the training time by reducing the total memristor number. The overall training time of 2-D reduced design is only 3.3% of that of the original design. When crossbar size rises, longer time is consumed on IR-drop compensation because of the severer voltage degradation. For instance, compensation overhead contributes to 4.12% of total training time when $n = m = 32$, and 28.3% when $n = m = 128$.

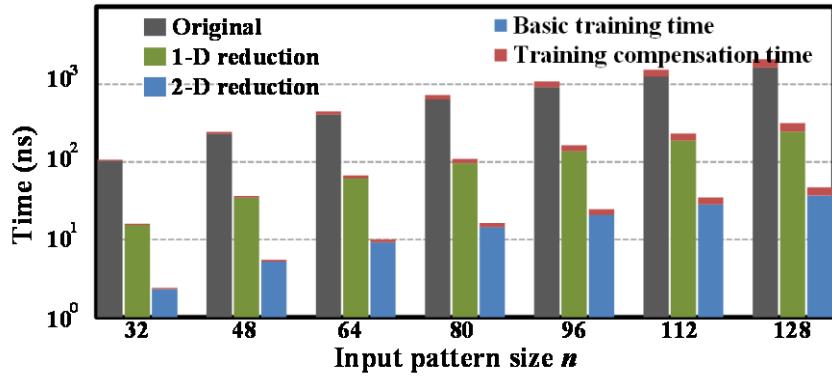


Figure 31. Training time comparison.

5.5.4 Area

As discussed in Section 5.2.3, system reduction scales down the size of crossbar while introducing additional peripheral circuit. We evaluate the overall circuit area cost of original, 1-D and 2-D reduced NCS designs, as shown in Figure 32. The circuit design details are also illustrated in Table 3. 1-D reduced design always has a smaller area than original design until $n = 96$, beyond which the overhead of extra circuit starts to dominate. 2-D reduced design, however, always has the smallest area: when $n = 128$, the area of 2-D reduced design is only 61.3% of that of original design. Note that the areas cost shown in Fig. 20 is for only a single crossbar and its peripheral circuit. When the NCS is scaled up to a level capable of processing large data size, e.g., high resolution image, multiple crossbar may be needed. Routing and analog data transmission will occupy significant portion of the circuit area.

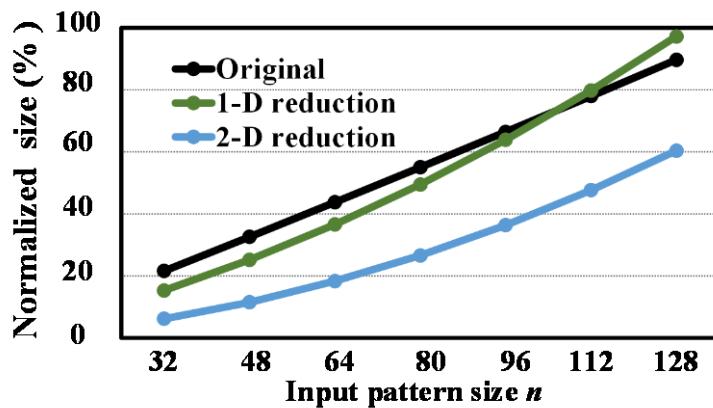


Figure 32. Area cost comparison.

5.5.5 Robustness

Similar to Section 5.5.1, we set the number of standard patterns stored in a crossbar to $0.15 n$ in our recall robustness analysis on the circuit implementation from Section 5.2.3. Each standard pattern is a randomly-generated binary vector. Test input patterns are the defected standard patterns where each digit has a 15% probability to be inverted. We determine whether a recall is successful by comparing the mismatch between the outputs of the test inputs and the corresponding standard patterns. Recall successful rate (testing accuracy) is obtained by running 1000 times Monte-Carlo simulations. Besides the process variations of memristor devices ($\sigma_m = 0.2$), we also assume each memristor has 0.1% chance to be stuck at HRS or LRS.

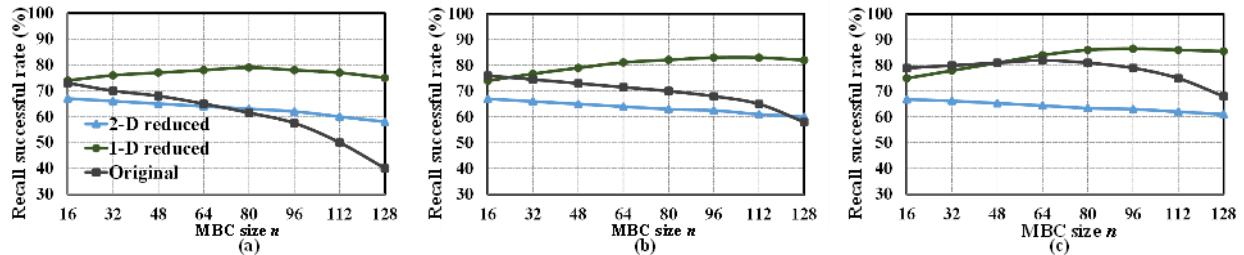


Figure 33. Recall successful rates of three NCS designs considering IR-drop.

5.5.5.1 Training and Testing with IR-drop

We first evaluate the NCS performance under the impact of IR-drop. The same experiment setup and training method in Section 5.5.2 are adopted in these simulations. Figure 33(a) shows the recall successful rate of three NCS designs when IR-drop is considered during both training and sensing process. Conventional NCS design suffers from the largest degradation

among all the designs when the scale of the crossbar increases from 16×16 to 128×128 . And 1-D reduction outperforms the other two designs.

Figure 33 (b) shows the results when IR-drop compensation is introduced during only training process. When the size of the crossbar increases, the training quality of the original crossbar is significantly improved by IR-drop compensation.

Figure 33 (c) shows the recall successful rate of all NCS designs when the IR-drop compensation is applied during both training and testing. IR-drop compensation substantially enhanced the robustness of original and 1-D reduced designs. As the crossbar size of all 2-D reduced designs is smaller than 20×20 , neither the IR-drop effect nor the compensation improvement is significant. When $n = m = 128$, the 1-D reduced design shows a recall successful rate of 85.3%, which is 27.0% higher than original design (68.3%).

5.5.5.2 Impact of memristor/wire resistance variation

In neural network model development, hardware device variations, e.g., memristor variation and metal wire resistance variation are generally not considered. In our IR-drop compensation design, wire resistance is also considered as a fixed value. In reality, these variations may harm the robustness of both training and recall process. Based on the experiment setup in section 5.5.5, we simulated the impacts of memristor and metal wire resistance variations and IR-drop. Table 4 shows the simulated recall successful rates of the NCS with different crossbar sizes and memristor variation assumptions. When the memristors have lower variation (i.e., $\sigma_m = 0.1$), all three designs have better recall successful rates. However, the recall successful rate of the 2-D reduced crossbar degrades more phenomenally than the other

two designs when σ increases, implying less tolerance to process variations as discussed in Section 5.2.4.

Although wire resistance greatly affects the impact of IR-drop, its variation σ_{wire} does not show visible impact on overall system robustness due to its relatively small magnitude (2.5 Ω) and variance ($\sigma_{\text{wire}} = 0.05$).

5.5.5.3 Tradeoff between 1-D/2-D system reduction

It is clear that among all the designs, 2-D reduced design has the best area efficiency even though it may not offer the same computation reliability as the 1-D reduced and conventional designs. However, computation reliability of 2-D reduced design shows higher sensitivity to the variation of memristor resistance (σ_m) than that of other two designs, as shown in Table 4.

Hence, 2-D reduced design is a good solution for a large-scale data processing with well-controlled variability of memristor device as well as the relatively low computation accuracy per iteration.

1-D reduced design offers a good balance among area efficiency, computation accuracy, and tolerance to IR-drop and memristor variations. In fact, 1-D reduced design even shows better computation accuracy than the original design when the problem size is large because of the significantly improved tolerance to IR-drop and memristor variations, as shown in Figure 33. However, as shown in Figure 32, the 1-D reduced design will lose its area benefit when $n \approx 100$ due to the increased overheads of pre-processing circuit.

Table 4. Recall successful rate of NCS with different sizes

σ_{wire}	σ_m	Reduction scheme	Stored patterns/Sizes			
			5/32	10/64	15/96	20/128
0	0.1	Original	0.934	0.902	0.835	0.716
		1-D reduction	0.917	0.933	0.958	0.944
		2-D reduction	0.884	0.875	0.844	0.833
	0.2	Original	0.803	0.822	0.797	0.683
		1-D reduction	0.788	0.844	0.868	0.853
		2-D reduction	0.663	0.644	0.628	0.607
0.05	0.1	Original	0.924	0.912	0.828	0.725
		1-D reduction	0.919	0.925	0.972	0.934
		2-D reduction	0.894	0.873	0.841	0.828
	0.2	Original	0.805	0.829	0.784	0.674
		1-D reduction	0.799	0.844	0.858	0.847
		2-D reduction	0.658	0.653	0.621	0.614

5.5.6 Model compression

In this experiment, we will demonstrate the hardware resource reduction by utilizing the model compression technology. The data set we use is the “LETTER” from UCI. Each sample contains a feature vector with length of 16 features, and every sample belongs to one of two classes. Every feature is a real value between 1 to 16. The original training set and testing set both contain 5k samples.

We generate 20k pseudo samples by randomly increase/decrease value of original samples with 10% probability. In this experiment we use SK-learn to build the ensemble model that contains 5 support vector machines, 5 random forests, 5 nearest neighbors and 5 decision trees as base models. The x-axis is the amount of memristor connections we use in the neural network and y-axis is the testing accuracy of the model. The accuracy of the best ensemble model, i.e., 4.58% error rate, does not change with the x-axis because it is not neural network. The blue line

is the neural network that is trained in the conventional way. The orange/gray line is the neural network trained with model compression methods with original/pseudo training samples. The difference between the original sample and pseudo samples is negligible in this experiment while the model compression technology is clearly demonstrated to significantly improve the performance of the neural network compared with conventional training. For example, it shows that, to achieve the sample performance, i.e., 7.12% error rate, the model compression technology can reduce the number of memristors by 75% (from 8704 memristors to 2176 memristors).

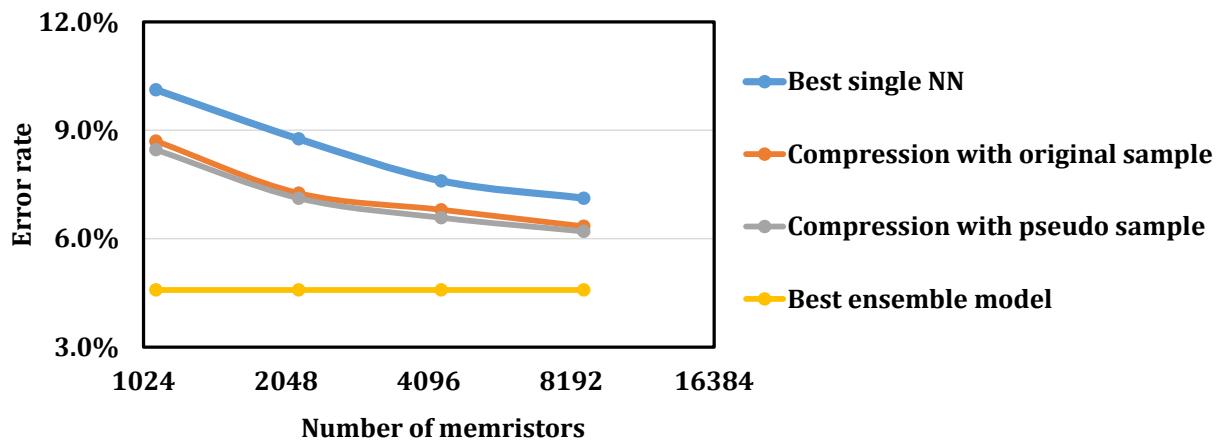


Figure 34. Model compression.

5.6 SECTION SUMMARY

In this section, our analysis reveals that the IR-drop along the metal wires and memristor arrays in crossbars significantly affects the reliability and scalability of the NCS. Based on our analysis, we proposed system reduction schemes that can substantially reduce the size of the crossbar required in NCS implementation for system robustness enhancement. We also proposed IR-drop compensation technique that can improve the training and recall reliability of the NCS. Simulations show that these techniques substantially improve the operation robustness of the NCS by 27.0% and reduce 38.7% of circuit area. Orthogonal to both of these two techniques, we also investigate utilizing model compression technique to reduce the hardware resources while achieve high performances, e.g., testing accuracy.

6.0 SECURITY APPLICATION

Cognitive systems that leverage advanced data processing technologies, e.g., machine learning and data mining, are widely used in various fields. The common goal of these data processing technologies is the automatic or semi-automatic analysis of large quantities of data, often to extract previously unseen patterns such as groups of data records (clustering) and unusual records (anomaly detection) [78].

Security of any data processing technology can be critical to users. The unique properties of a cognitive system will also introduce new security challenges. For example, a user may need to use powerful computing sources from a third party for data processing. Hence, [75] investigated protecting the privacy of data that needs to be processed by a third party. In this work, we will explore other security concerns related to the basic components of cognitive systems: the training data and processing model.

While the training data are often confidential because they are hard to collect or sensitive to the public, the features of the training data can be obtained by anyone with access to the cognitive system as a normal user. Further, although the processing model of the cognitive systems can be highly valuable to the owner, we demonstrate that it can be replicated by a skilled attacker, even without any information about the model selection or the original training data.

Based on our understanding of security concerns of the cognitive systems, we investigate the possibility of utilizing the unique property of memristor device to build a secured

neuromorphic computing system that prevents replicating the model hard-coded in a memristor-based terminal device by leveraging memristor's drifting effect. The performance of the system will gradually degrade without regular calibrations. Experimental results show that our design can effectively provide resilience to replication attack.

6.1 SECURITY CONCERNS IN COGNITIVE SYSTEMS

6.1.1 Test Data Privacy

The user/testing data that needs to be processed and can be very sensitive. For example, a hospital may want to predict a patient's risk of certain disease. However, due to regulations, the patient's medical files cannot be revealed. If the hospital wish to use popular cloud machine learning (CML) services from commercial providers' infrastructures, e.g., Microsoft Azure Machine Learning, Google Prediction API, Amazon, etc., the privacy of the patient's medical files is under serious thread. This particular problem is searched by P. Xie [75]. They proposed a neural network model that allows user to encrypt test data before sending to the model. The model can generate prediction results that are also encrypted and send back to the user. However, the complexity of the model they use is restricted.

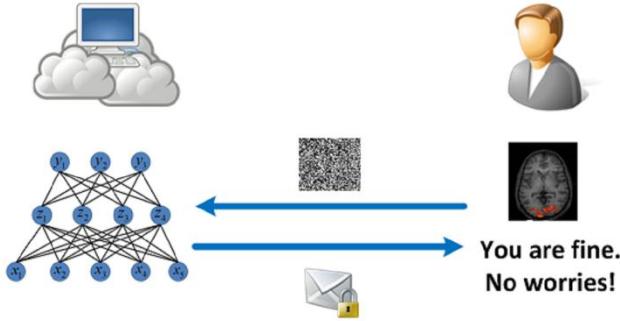


Figure 35. Encrypted neural network.

6.1.2 Training Data Security

Training data can be sensitive in many cognitive systems and becomes a major security concern. For instance, in a supervised learning based classification application, e.g., a surveillance system, the target to be searched is often sensitive information that should not be released to the public [79]. However, the security of the searching target (training data) may be threatened if the classifier itself is available to the users, e.g., the terrorists obtained a surveillance systems that can be tested. The concerned problem can be formulated as follows:

Without losing generality, we use classification model as an example and describe function of the considered cognitive system as:

$$g(\mathbf{w}, \mathbf{x}) = p(\mathbf{y} = y_i | \mathbf{w}, \mathbf{x}), i = 1 \dots n \quad (40)$$

Here, \mathbf{w} is a vector of the parameters of the original model. \mathbf{x} is the input vector of features. In equation (40), y_i is the i -th class that a sample can be assigned to. The probability function

$p(y=y_i|\mathbf{w}, \mathbf{x})$ is defined by the structure of the original model, e.g., neural network or naïve Bayesian model.

When a training data set \mathbf{D} is given ($D_i = \langle x_i, y_i \rangle | i=1..r$), the goal of the training is to find a \mathbf{w} so that:

$$\max_{\mathbf{w}} L(\mathbf{D}, \mathbf{w}) = \max_{\mathbf{w}} \prod_{i=1}^r p(y = y_i | x_i, \mathbf{w}) \quad (41)$$

After training, the optimal \mathbf{w} will be selected.

Here we assume that the attacker can access the classifier model function, i.e., $g(\mathbf{w}, \mathbf{x})$, but cannot access the original training data set \mathbf{D} . The attacking procedure to “reverse-engineer” the searching target of the original training data can be then translated to finding a data set \mathbf{x}' to achieve the maximum probability of the model function: $\max_{\mathbf{x}'} g(\mathbf{w}, \mathbf{x}')$. The procedure of

searching \mathbf{x}' can be as simple as the standard gradient-descent algorithm with random starting point in the feature space. To demonstrate this theory, we use MNIST hand writing digits from 0 to 9 as the training data. The model used for classification is a neural network. The training function of the classifier is to differentiate number ‘0’ from other digits. The standard gradient-descent method is used in the training procedure.

Single-layer neural network: The function of a single-layer neural network can be described as:

$$\mathbf{y} = g(\mathbf{x}) = \text{sigmoid}(\mathbf{w} \cdot \mathbf{x}), \quad (42)$$

where \mathbf{x} is the input feature vector and \mathbf{w} is connection vector. After finishing the training process, we try to find the input pattern \mathbf{x}' that generates the maximum output, such as:

$$\max_{\mathbf{x}'} \text{sigmoid}(\mathbf{w} * \mathbf{x}'). \quad (43)$$

Here, each element in \mathbf{x} is limited within the range between 0 and 1, representing the brightness value of a pixel. Because the sigmoid function is monotonic increasing, a close form solution of \mathbf{x}' exists as follows:

$$x'_i = \begin{cases} 1, & \text{if } w_i > 0 \\ 0, & \text{if } w_i \leq 0 \end{cases}. \quad (44)$$

Accordingly, \mathbf{x} represents a binary image as shown in Figure 36 (b).

Multi-layer neural network: The situation for a multi-layer neural network will be much more complicated. Since nonlinearity will be introduced to the classifier, we cannot derive a close form solution. Searching for the input pattern that generates max output probability will be determined by the gradient descent algorithm. Figure 36 (c) shows the input pattern \mathbf{x}^* that generates highest probability, which matches the original searching target very well.

In summary, for both single- and multi-layer neural networks, the searching target can be reversely estimated through the classification model.

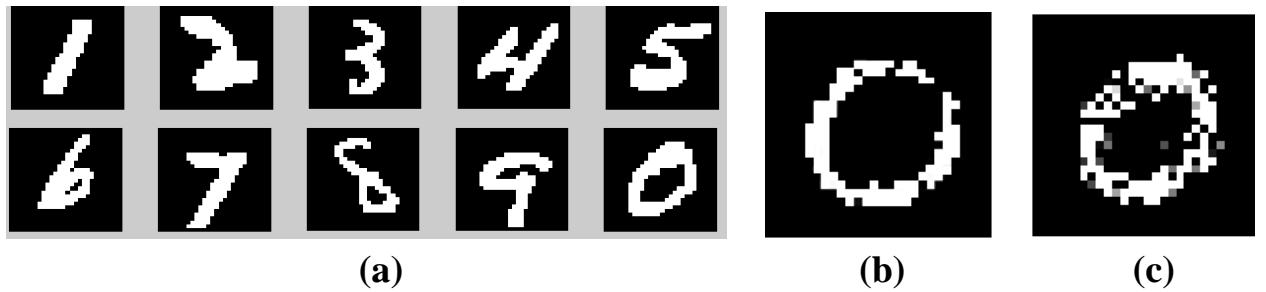


Figure 36. Reverse estimation.

6.1.3 Model Security

A lot of models are carefully designed and very valuable to the owners. At the same time, running learning models on an embedded device introduces an obvious convenience such as run-time processing and high efficiency, but unfortunately also introduces security challenges. The learning model will be exposed to the risk of being attacked by unauthorized attackers who have physical access to the device. Consider the following scenario: Assuming there is a drone carrying an image processing system, which is being used for its navigation and guidance systems. If the drone is captured by an unauthorized third party, say, an attacker, the attacker may use the in-put/output (I/O) pairs obtained from the drone to “learn” the function implemented by the system [80]. This scenario is similar to most side-channel attacks, where the attacker tries to learn a secret key embedded in the device to which he has physical access. This type of security concern interests us most and we will investigate potential hardware solution.

6.2 MODEL REPLICATION ATTACK

6.2.1 Attacking Model

Figure 37 shows a conceptual view of the concerned embedded system and its usage model. The model is first trained for an application and then the user can submit his/her data for processing (testing), e.g., pattern recognition or classification.

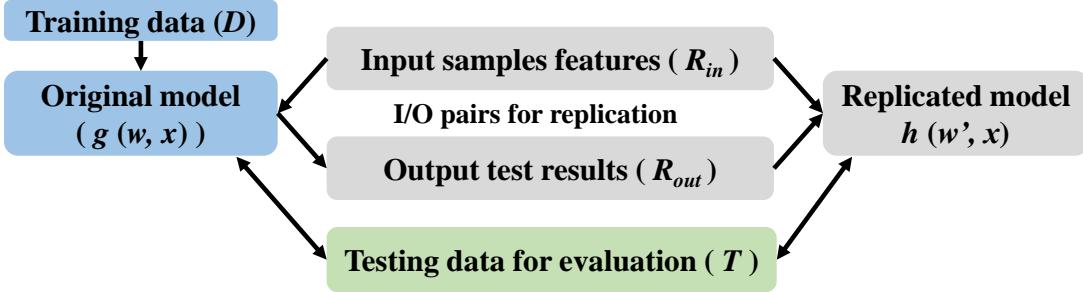


Figure 37. Training and replication of the learning model.

We still use the function defined in equation (40) to describe the original learning model $g(\mathbf{w}, \mathbf{x})$, which is used to solve a classification problem. To define the attack process realistically, we summarize the accessibility of the attacker as below:

- The attacker can submit test samples, e.g., images, body data from patients, finger prints, to the originally trained model and receive the test results without any constraints, i.e., being granted with the same privilege as a normal user or being able to physically get access to it;
- The attacker does not have access to the original training data;
- The attacker has no knowledge about the info and parameters of the original model.

The goal of the attacker is to replicate the function of the original model $g(\mathbf{w}, \mathbf{x})$ by constructing a new model $h(\mathbf{w}', \mathbf{x})$. To achieve this goal, one of the possible attack process may consist of the following two phases, namely, model testing and model replicating.

Model Testing: The above accessibility assumption limits the interactions between the attacker and the NCS to only testing the data as a normal user. Therefore, the only information that the attacker may obtain (indirectly) about the original model is through submitting testing data samples and collect output from the NCS. In this paper, we use D_{in} to denote the testing samples, each of which contains a vector of features requested by the NCS to make a prediction. D_{out} is the predicted results generated by the NCS. $[D_{in}, D_{out}]$ construct I/O pairs. Here the length of D_{in} , i.e., the number of testing samples, and the length of D_{out} and $[D_{in}, D_{out}]$ are the same, say, m, which is decided by the attacker. We refer to the above process of constructing the I/O pairs as model testing.

Model Replication: Since the attacker has no knowledge about the type of the original model, he/she needs to select a learning model as a starting point for model replicating. After the I/O pairs are constructed and the replicated model type is selected, the attacker starts to use the I/O pairs to training the replicated model: since the I/O pairs are generated from the original model, the function of the replicate model will gradually approach to that of the original model during.

The model selected by attacker is arbitrary. Besides the original model, (e.g., neural network), we could also use other model (e.g., Naïve Bayes model) as the replicated model to learn the function of the original model. In addition, it has been proved that it is not necessary to select the same model type as the one of the original model [15]. An important evaluation criterion of a NCS is testing accuracy. Usually when a NCS is trained, a set of data samples with ground-truth labels are used to evaluate its accuracy, which is defined in our paper as:

$$\text{testing accuracy} = \frac{\text{number of true-positives}}{\text{number of all testing samples}} \quad (45)$$

Here the number of true-positives is the number of predictions that match the ground-truth labels. In addition, we define the replication quality as the difference between the testing accuracy of the replicated model and the original model. The smaller the difference is, the better the replication quality will be.

6.2.2 Demonstration

In this case study, we adopted the well-known PIMA Indians Diabetes Data Set for diabetes prediction [81]. The data set collects many patients' medical records (features) including age, weight, blood pressure, pregnancy, result of glucose tolerance test, etc. Here, we use neural network as the original model. The original training set consists of 500 training samples, each of which contains eight features and one label. The label could be either 1 (positive) or 0 (negative). Training is finished when the model can successfully make prediction for 80.1% of the training samples. During the testing stage, we apply another 229 samples and the neural network model diagnoses 79.3% of them correctly.

Assume that attackers have obtained features of a new group of patients and sent these test samples to the original classifier. By combining these test samples and results, attackers can form “I/O pairs” for the model replication. Due to the lack of real patient data, in the case study, we randomly generate a set of patient features based on the distribution of certain features.

The size of the I/O pairs is an important factor. Intuitively, a large set of I/O pairs potentially makes the replicated model easier and more accurate. However, a large set of training data also indicates higher costs in the data set generation and the model replication. The attacking will become invalid and meaningless if the replication is not much less expensive than creating another “original” model.

In our experiments, we vary the size of the I/O pairs from 10 to 600 and test the accuracy of the replicated models. For the holdout test, a small set of I/O pairs is sufficient to develop a replication model with almost the same test rate as the original classifier, as illustrated in Figure 38. We use neural network and naïve Bayesian models as the alternative model selection. Our experiments demonstrate that the type of the replication model does not affect much on the predication accuracy.

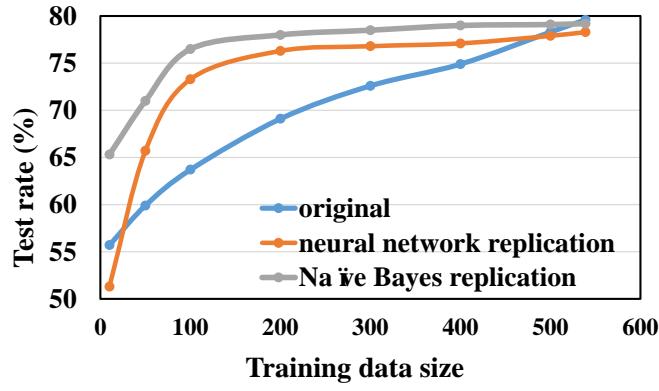


Figure 38. Model replication.

6.3 MEMRISTOR-BASED SECURED NCS

6.3.1 Drifting Effect

The drifting effect of a memristor denotes the fact that the resistance of the device will gradually change after being programmed. This drifting effect has two major contributors: 1) the intrinsic retention property of the device [82], and 2) the read-induced change in resistance.

Retention property denotes the fact that the resistance of a programmed memristor device continuously changes to a high resistance state without any applied voltage. This type of resistance change is very slow and hard to control since it is related to the material relaxing mechanism.

During sensing, a memristor device is constantly stimulated by short minor voltage pulses, and its resistance keeps changing. Based on switching behavior described in Figure 2, the drifting rate (i.e., changing rate of its resistance) can be controlled by choosing the amplitude and duration of the sensing current/voltage. In general, the resistance change of a memristor in a memristor-based hardware is a continuous procedure that can be described as:

$$\Delta R = f(v, t). \quad (46)$$

Here, v and t are the sensing voltages and operation time of the memristor, respectively.

6.3.2 Secured NCS Design

Since the elements in the weight matrix \mathbf{W} of a neural network can be either positive or negative but resistances of memristors can be only positive, we partition \mathbf{W} into two matrices \mathbf{A} and \mathbf{B} as:

$$a_{ij} = \begin{cases} w_{ij}, & \text{if } w_{ij} > 0 \\ 0, & \text{if } w_{ij} \leq 0 \end{cases} \quad (47)$$

$$b_{ij} = \begin{cases} 0, & \text{if } w_{ij} > 0 \\ -w_{ij}, & \text{if } w_{ij} \leq 0 \end{cases} \quad (48)$$

Here $w_{ij} \in \mathbf{W}$ denote the elements in \mathbf{W} . Matrices \mathbf{A} and \mathbf{B} are represented using one memristor

crossbar for each (\mathbf{M}_1 and \mathbf{M}_2 , respectively) where the conductance of every memristor "g>0".

As such, equation (1) can be rewritten as:

$$\mathbf{y}_n = f(\mathbf{x}_m \cdot \mathbf{A}_{m,n} \cdot \mathbf{x}_m \cdot \mathbf{B}_{m,n}). \quad (49)$$

For simplicity, here we assume the resistance change of memristors is a linear process. In conventional design, same inputs are applied to both \mathbf{A} and \mathbf{B} during computation. If we use $\Delta\mathbf{A}$ and $\Delta\mathbf{B}$ as the value drifting of \mathbf{A} and \mathbf{B} , then the drifting of \mathbf{W} can be expressed as $\Delta\mathbf{W}=\Delta\mathbf{A}-\Delta\mathbf{B}$.

According to equation (47) and equation (48), we have:

$$\Delta\mathbf{W} = \mathbf{W} \circ \text{sign}(\mathbf{W}) \cdot \text{rate} \cdot \text{input}. \quad (50)$$

Here “◦” denotes the Hadamard product of two matrices. *rate* is a coefficient indicating the drifting rate and *input* is application specific.

As the resistance of the crossbar changes through usage, the accuracy of the system degrades over time, which means the function of the model behind the system is changed. We

refer to this property of secured NCS as its forgetting property. In order to better control this property, we propose to apply random voltage pulses to all memristors for each I/O pair, so that the resistance can change linearly.

In a replication process, if the I/O pairs are generated by this gradually degraded system, the highest testing accuracy that can be achieved by the replicated model will be significantly lower than the one generated using the ideal I/O pairs. This testing accuracy difference demonstrates the potential of using the forgetting property to overcome the model replication challenge.

In order to guarantee the usability for normal users, a calibration mechanism must be applied to such a system with forgetting property. A naïve way to do calibration is to refresh the crossbars with initial resistance states periodically, using a secured method. For example, in the scenario mentioned in Section 6.1.3, owner of the drone could calibrate it through a secured communication channel from the base. This can be performed through authentication protocols using physical unclonable functions [83][84].

6.4 EXPERIMENT RESULTS

In this section, we will demonstrate the effectiveness of memristor-based secured NCS described in Section 6.3.2. In Section 6.4.1, we will depict the drifting effect of memristor devices and its relationship to the testing accuracy of the system. In Section 6.4.2, we will show the protection effectiveness against replication attack. In the experiments, we choose a benchmark from Scikit-learn [85]: Hand-written Digits (Digit).

6.4.1 Drifting vs. Degradation

The existence of drifting effect causes the function of a secured NCS gradually deviates from the trained one during its operation. To evaluate the impact of drifting effect on the testing accuracy of a NCS, we simulate the degradation brought by the drifting in such a system when running different benchmarks. The memristor crossbar is configured to implement a neural network with two hidden layers for all benchmarks. Each layer has 64 neurons.

The simulation results are summarized in Figure 39. Here, we take *Digit* for example, as the curves for all three benchmarks have very similar trend. X-axis denotes the number of I/O pairs. The left y-axis denotes the error rate of the system while the right y-axis denotes the normalized summed absolute changes of weights (NSCW) due to memristor drifting. We define the error rate and NSCW as:

$$\text{error rate} = \frac{1}{n} \sum_{i=1}^n \text{if}(y_i == t_i). \quad (51)$$

$$\text{NSCW} = \sum_{i,j} |w_{ij} - w'_{ij}| / \sum_i |w_{ij}|. \quad (52)$$

y_i denotes the classification result, t_i denotes the target result, n denotes the size of I/O pairs, and w_{ij} is the element of weight matrices in neural network. Our results show that with increasing resistance change of memristor device, the testing accuracy drops significantly to unusable level (error rate > 60%).

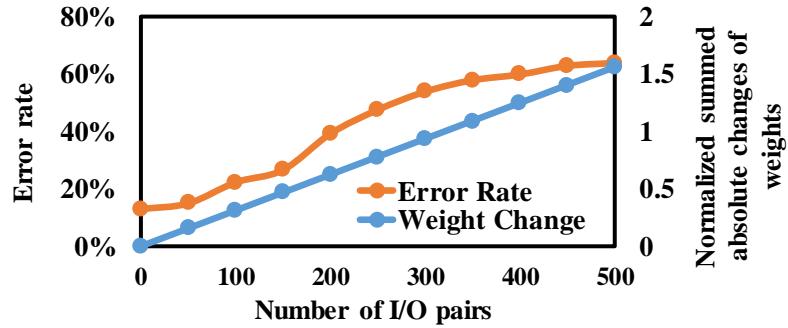


Figure 39. Resistance change and system degradation.

6.4.2 Replication Quality

In this section, we will show the effectiveness of our design on preventing replication attack. The effectiveness is evaluated by showing the highest testing accuracy that can be achieved by a replicated model. Lower testing accuracy means the system has better resilience against replication attack.

Figure 40 summarizes the results of comparison, where x-axis is the I/O pairs and y-axis is the accuracy rate. The model chosen for replication is the model with the best testing accuracy, e.g., SVM for Digit, Random Forest for Faults. Other models include K-Nearest Neighbors and neural network. In the simulation, we assume the attacker uses the original training samples to train their replication model, by doing so, the attacker will get a better quality than using manually generated samples.

Figure 40 shows the testing accuracy on degraded system is similar to the original one at the beginning. That is because it belongs to the low error rate region as we can observe from the curve in Figure 39. For the secured NCS, the accuracy of replicated system drops after 150 I/O pairs, while the accuracy of original model increases. This case study shows the proposed secured NCS design is resilient to replication attack.

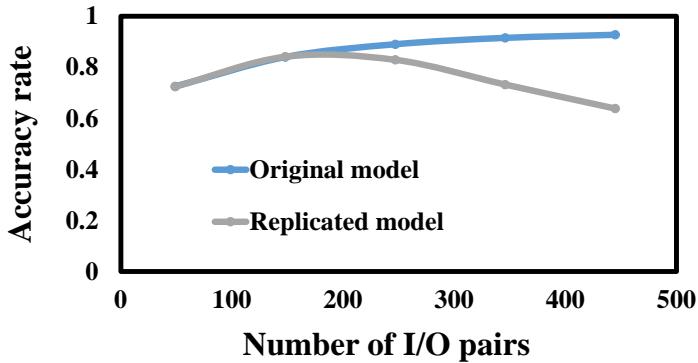


Figure 40. Effectiveness of memristor-based secured neuromorphic system.

6.5 SECTION SUMMARY

In this section, we first introduced some security concerns of modern cognitive systems. Several application cases are presented to demonstrate that the features of the private training data and the confidential data processing model may be obtained and replicated by attackers granted with the access at only normal user level. Our case study shows that with only 37% of the original training data size, the replicated system can achieve up to 95% of the test rate of the original system. In addition, we investigated using unique drifting property of memristor to build a secured NCS that protects the processing model from replication attack. Our experiments demonstrates the effectiveness of our proposed solution.

7.0 CONCLUSION AND FUTURE WORK

7.1 DISSERTATION CONCLUSION

Neural network, as an important type of machine learning algorithms, has been widely used in various fields of data processing applications. Many embedded hardware platforms, including FPGA and System-on-Chip (SoC), have been developed to implement neural networks with excellent speed and efficiency. Recently, the similarity between the programmable resistance state of memristor devices and the variable weight connection in neural networks dramatically simplifies the structure of circuit realization of a neural network. However, constraints and imperfections of hardware circuit severely limit the memristor-based NCS from achieving high reliability and high efficiency at the same time. This dissertation quantitatively investigated the testing accuracy, training speed and hardware cost of memristor crossbar based NCS when considering hardware limitations, e.g., device variation, analog signal noise and IR-drop. Based on our understanding, we proposed three main techniques:

In section 3.0 , we proposed “Vortex” – a variation-aware off-device training scheme that tolerates device imperfections and design constraints. By combining variation-aware training and adaptive mapping method, we demonstrate that Vortex achieves significantly improved training quality, i.e., 29.6% higher test rate, w.r.t. conventional off-device training.

On-device training has unique advantage of combat the impact of device variation because the training monitoring the output of the devices iteratively. Hence, in section 4.0 , we proposed a noise-eliminating on-device training method and digital-assisted initialization step to improve the training process robustness and the speed of on-device training. Experimental results show that our techniques can significantly improve the testing accuracy and training time of neuromorphic computing system by up to 18.7% ~ 36.2% and 12.6% ~ 14.1%, respectively, through suppressing the noise accumulation in the training iterations and reducing mismatch between the initial weight matrix state and the target value.

Besides training quality and speed, scalability of memristor based NCS is another very critical factor that defines the hardware usability and efficiency. In section 5.0 , our analysis reveals that the IR-drop in crossbars significantly affects the reliability and scalability of the NCS. Then we proposed hardware solutions, e.g., system reduction and IR-drop compensation schemes that can significantly reduce the size of the crossbar required in NCS and improve the training and testing reliability of the memristor crossbar. From a different approach, we also investigate utilizing model compression technique to reduce the hardware resources while achieve high performances on the software level. The software and hardware solutions are orthogonal to each other and can be integrated to be a systematic solution.

In section 6.0 , we first introduced some security concerns, including private data and processing model safety, of modern cognitive systems. We investigated using the unique drifting property of memristor to build a secured NCS that protects the processing model from replication attack. Our experiments demonstrates the effectiveness of our proposed solution.

7.2 FUTURE WORK

7.2.1 Function Generality of Memristor-based NCS

So far, we only demonstrated fully connected neural network on memristor-based NCS.

To achieve generality, we need a mechanism that enables programmable neural topology for various neural network model, e.g., recurrent neural network and convolutional neural network.

The recurrent operation can be easily implemented since the output of one crossbar can be sent back to the input of any crossbar, including itself. The convolution operation can also be realized by carefully mapping the convolutional filter onto the crossbar. As shown in Figure 41, a typical convolutional operation applies a local filter $F_{l \times k}$ on different area of a feature map $M_{(n,r)}$.

Without losing generality, we only use the first column, i.e., $M_{(n,l)}$, of the map as an example.

The output of the convolution operation can be described as:

$$o_i = F_{(1,k)} \cdot M_{(i:i+k,1)} \quad (53)$$

In a crossbar, we can program the memristors in a way such that in the i -th column, the memristor from the i -th to the $(i+k)$ -th row are programmed to represent $F_{(1,1:k)}$, while the other memristors set to high resistance state. Obviously, output of the i -th column of the crossbar equals to the o_i in equation (53). By doing this, one channel (one filter) of convolutional operation can be successfully mapped onto one crossbar computing core. More implementation details need to be investigated to enable a hardware demonstration.

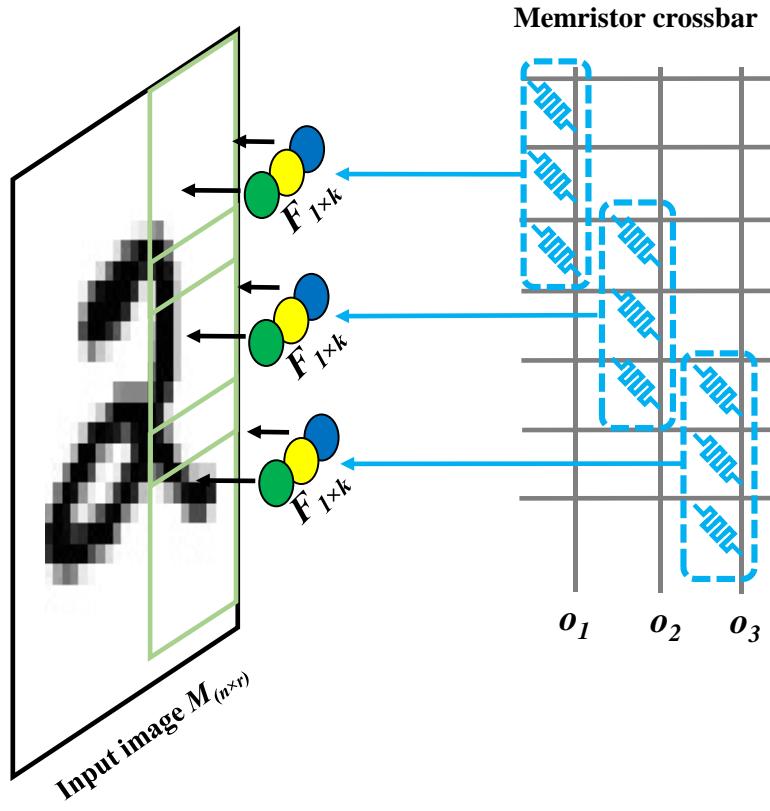


Figure 41. Memristor crossbar-based CNN.

7.2.2 Usability of Memristor-based Secured NCS

Maintaining a high testing accuracy of the original model, which means the resistance of memristor is changing in a low rate, offers better service quality to normal users, however, also makes the attacker replicate the model faster. The most ideal testing accuracy degradation curve is a step function that a controllable threshold value (shown as green curve in Figure 42). Before the threshold t , the system produces results with maximum accuracy to the normal user. At the same time, the system prevents the attacker from obtaining valuable information by sharply

reducing the testing accuracy after t . One goal of our future work can be described as improving the degradation curve from current red curve in to a more non-linear blue curve Figure 42.

The goal can be achieved from different approaches. In hardware circuit, we can try to identify critical paths that have the most impact on the testing output and connect them with specially designed memristor device. On software level, the neural network can be organized and trained to have extra feature that meets our requirement for better security.

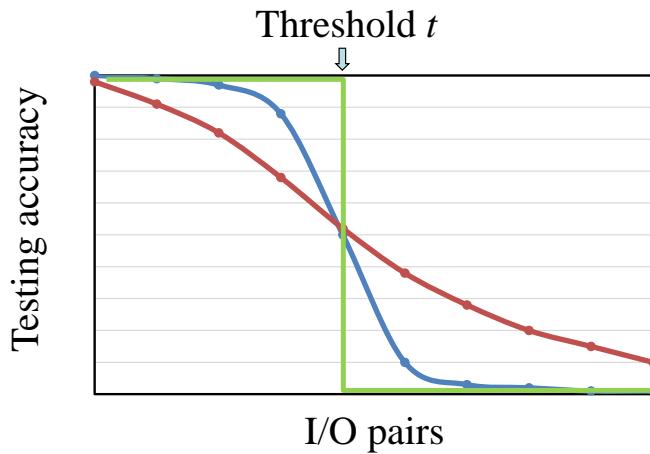


Figure 42. Ideal degradation.

BIBLIOGRAPHY

- [1] Y. Lecun, Y. Bengio and G. Hinton, “Deep Learning,” *Nature*, vol. 521, pp. 436–444, 2015.
- [2] S. Lawrence, C. L. Giles, Ah Chung Tsoi , and A. D. Back, “Face recognition: a convolutional neural-network approach,” *IEEE Transactions on Neural Networks*, pp. 98-113, 1997.
- [3] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Cernocky, “RNNLM - Recurrent Neural Network Language Modeling Toolkit,” *IEEE Automatic Speech Recognition and Understanding Workshop*, 2011.
- [4] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems* 2012.
- [5] R. Dolan, G. DeSouza, “GPU-based simulation of cellular neural networks for image processing,” *International Joint Conference on Neural Networks*, 2009.
- [6] A. A. Mahesri, and V. Vardhan, “Power Consumption Breakdown on A Modern Laptop,” *International Power Aware Computer Systems Workshop*, 2004.
- [7] Kaya, and A. R. Brown, “Intrinsic Parameter Fluctuations in Decanometer MOSFETs Introduced by Gate Line Edge Roughness,” *IEEE Transaction on Electron Devices*, vol. 50, pp. 1254–1260, 2003.
- [8] Y. Chen, “Spintronic Devices – An Alternative Path to Continue Moore’s Law,” *Dalian Mini-Colloquia of IEEE Electron Device Society*, Jun. 2010.
- [9] R. Nebashi, N. Sakimura, Y. Tsuji, S. Fukami, H. Honjo, S. Saito, S. Miura, N. Ishiwata, K. Kinoshita,T. Hanyu, T. Endoh, N. Kasai, H. Ohno, and T. Sugibayashi, “A Content Addressable Memory Using Magnetic Domain Wall Motion Cells,” *Symposium on VLSI Circuits*, pp.300 –301, 2011.
- [10] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, “Spintronic Memristor Through Spin-torque-induced Magnetization Motion,” *IEEE Electron Device Letters*, vol. 30, pp. 294–297, 2009.

- [11] Y. V. Pershin and M. D. Ventra, “Spin Memristive Systems: Spin Memory Effects in Semiconductor Spintronics,” *Physical Review B*, vol.78, pp.159905, 2008.
- [12] P. Wang, W. Zhang, R. Joshi, R. Kanj, and Y. Chen, “A Thermal and Process Variation Aware MTJ Switching Model and Its Applications in Soft Error Analysis,” *International Conference on Computer Aided Design*, Nov. 2012.
- [13] L. O. Chua, S. M. Kang, “Memristive Devices and Systems,” *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209-223, 1976.
- [14] A. Bachtold, P. Hadley, T. Nakanishi, C. Dekker, “Logic Circuits with Carbon Nanotube Transistors,” *Science*, vol. 294, no. 5545, pp.1317-1320, 2001.
- [15] A. K. Geim, K. S. Novoselov, “The Rise of Graphene,” *Nature Materials* vol.6, pp.183-191, 2007. H. Chen, S. Saighi, et al, “Real-Time Simulation of Biologically Realistic Stochastic Neurons”, *VLSI. In IEEE Transactions on Neural Networks*, pages 1511-1517, 2010.
- [16] L. Chua., “Memristor-the Missing Circuit Element,” *In IEEE Transactions on Circuit Theory*, pages 507-519, 1971.
- [17] K. H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, “A Functional Hybrid Memristor Crossbar-Array/CMOS System for Data Storage and Neuromorphic Applications,” *Nano Letters*, vol. 12, no. 3, pp.389-395, 2011.
- [18] B. Q. Lai, L. Zhang, Z. Li, et al, “Ionic / Electronic Hybrid Materials Integrated in a Synaptic Transistor with Signal Processing and Learning Functions,” *Advanced Materials*, vol. 22, pp. 2448-2453, 2010.
- [19] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale Memristor Device as Synapse in Neuromorphic Systems,” *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.
- [20] B. T. Hasegawa, T. Ohno, K. Terabe, et al, “Learning Abilities Achieved by a Single Solid-State Atomic Switch,” *Advanced Materials*, vol. 22, pp.1831-1834, 2010.
- [21] Y. V. Pershin, M. D. Ventra, “Experimental Demonstration of Associative Memory with Memristive Neural Networks,” *Neural Networks*, vol. 23, no. 7, pp.881–886. 2010.
- [22] S. Wen, Z. G. Zeng, T. W. Huang, “Exponential Stability Analysis of Memristor-based Recurrent Neural Networks with Time-varying Delays,” *Neurocomputing*, vol. 97, pp. 233–240, 2012.
- [23] G. Snider, “Spike-timing-dependent Learning in Memristive Nanodevices,” *Memristor and Memristive systems Symposium*, pp. 85–92, 2008.
- [24] H. Kim, and M.P. Sah, and C. Yang, and T. Roska, and L.O. Chua, “Memristor Bridge Synapses,” *Proceedings of the IEEE*, vol.10, pp.2061-2070, 2012.

- [25] H. Wang, H. Li, and R. E. Pino, “Memristor-based Synapse Design and Training Scheme for Neuromorphic Computing Architecture,” International Joint Conference on Neural Networks, 2012.
- [26] K. D. Cantley, A. Subramaniam, H. J. Stiegler, et al, “Hebbian Learning in Spiking Neural Networks With Nanocrystalline Silicon TFTs and Memristive Synapses,” IEEE Transaction on Nanotechnology , vol. 10, no. 5, pp. 1066-1077, 2011
- [27] D. Strukov, J. Borghetti, and S. Williams, “Coupled Ionic and Electronic Transport Model of Thinfilm Semiconductor Memristive Behavior,” SMALL, vol. 5, pp. 1058–1063, 2009.
- [28] M. Hu, H. Li, Q. Wu, and G. Rose, “Hardware Realization of Neuromorphic BSB model with memristor crossbar network,” IEEE Design Automation Conference, pp. 554–559, 2012.
- [29] M. Hu, H. Li, Q. Wu, G. Rose, and Y. Chen, “Memristor Crossbar Based Hardware Realization of BSB Recall Function,” International Joint Conference on Neural Networks, 2012.
- [30] M. Hu, H. Li, Q. Wu, G. S. Rose and Y. Chen, “BSB Training Scheme Implementation on Memristor-Based Circuit,” IEEE Symposium Series on Computational Intelligence. Singapore, 2013.
- [31] F. M. Bayat, S. B. Shouraki, “Programming of Memristor Crossbars by using Genetic Algorithm,” Procedia Computer Science, vol. 3, pp. 232-237, 2011.
- [32] F. M. Bayat, S. B. Shouraki, and A. Rohani, “Memristor Crossbar-Based Hardware Implementation of the IDS Method,” IEEE Transaction on Fuzzy Systems, vol. 19, no. 6, pp.1083-1096, 2011.
- [33] S. H. Jo, K. H. Kim, W. Lu, “High-Density Crossbar Arrays Based on a Si Memristive System,” Nano Letters, vol. 9, no. 2, pp. 870-874, 2009.
- [34] L. Chen, C. Li, T. Huang, Y. Chen, X. Wang, “Memristor Crossbar-based Unsupervised Image Learning,” Neural Computing and Applications, 2013.
- [35] H. Chen, S. Saighi, L. Buhry, and S. Renaud, “Real-time Simulation of Biologically Realistic Stochastic Neurons in VLSI,” IEEE Transactions on Neural Networks, vol. 21, no. 9, pp. 1511-1517, 2010.
- [36] M. Sharad, D. Fan, and K. Roy, “Ultra Low Power Associative Computing with Spin Neurons and Resistive Crossbar Memory,” Design Automation Conference, Article no. 107, 2013.
- [37] M. Hu et al., “Hardware realization of BSB Recall Function Using Memrisotr Crossbar,” In DAC, pages 498-503, 2012.

- [38] S. H. Jo et al., “Nanoscale Memristor Device as Synapse in Neuromorphic Systems” In Nano Letters, pages 1297-1301, 2010.
- [39] K. Kim et al., “A Functional Hybrid Memristor Crossbar-Array/CMOS System for Data Storage and Neuromorphic ,” In Nano Letters, pages 389-395, 2012.
- [40] J. Klein et al., “Hight fault tolerance in neural crossbar,” In DTIS, pages 1-6, 2010.
- [41] R. Kumar, D. Tullsen, et al. “Heterogeneous chip multiprocessors,” In Computers, pages 32-38, 2005.
- [42] B. Liu, X. Li, T. Huang, Q. Wu, M. Barnell, H. Li, Y. Chen, “Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems,” International Conference on Computer-Aided Design (ICCAD), pp. 63-70, 2014.
- [43] B Liu, X Li, Q Wu, T Huang, H Li, Y Chen, “Vortex: variation-aware training for memristor X-bar,” Design Automation Conference (DAC), pp. 1-6, 2015.
- [44] B Liu, C Wu, Q Wu, M Barnell, Q Qiu, H Li, Y Chen, “Cloning your mind: security challenges in cognitive system designs and their solutions,” invited to Design Automation Conference (DAC), pp. 95, 2015.
- [45] B Liu, Y Chen, B Wysocki, T Huang, “Reconfigurable neuromorphic computing system with memristor-based synapse design,” Neural Processing Letters, vol. 41, pp. 159-167, 2015.
- [46] Xiaoxiao Liu, Mengjie Mao, Beiyi Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, Jianhua Yang, “RENO: a high-efficient reconfigurable neuromorphic computing accelerator design,” Design Automation Conference, pp. 1-6, 2015.
- [47] F. Moreno et al., “Reconfigurable Hardware Architecture of a Shape Recognition System Based on Specialized Tiny Neural Networks With Online Training” In IEEE Transactions on Industrial Electronics, page 3253-3263, 2009.
- [48] S. Yu, Y. Wu and P. Wong., “Modeling the Switching Dynamics of Programmable-Metallization-Cell (PMC) Memory and its Application as Synapse Device for a Neuromorphic Computation System” In APL, page 103514-3, 2011.
- [49] T. Yu et al., “Analog VLSI Biophysical Neurons and Synapses With Programmable Membrane Channel Kinetics” In IEEE Transactions on Biomedical Circuits and Systems, page 139-148, 2010.
- [50] F. Zhuo and P. Li, “Multigrid on GPU: Tackling Power Grid Analysis on parallel SIMT platforms” International Conference on Computer-Aided Design (ICCAD), pages 647-654, 2008.

- [51] Asenov, A, et al., “Intrinsic Parameter Fluctuations in Decanometer MOSFETs Introduced by Gate Line Edge Roughness”, IEEE Transaction on Electron Devices, 2003.
- [52] X. Dong, et al., “Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement”, DAC, 2008.
- [53] D. Niu, et al., “Impact of process variations on emerging memristor”, DAC, 2010.
- [54] A. Sally, “Reflections on the Memory Wall,” CCF, 2004.
- [55] M. Sharad, et al., “Ultra Low Power Associative Computing with Spin Neurons and Resistive Crossbar Memory”, DAC, 2013.
- [56] M. Hu, et al., “Hardware realization of BSB recall function using memristor crossbar arrays”, DAC, 2012.
- [57] J. Liang, et al., “Cross-Point Memory Array Without Cell Selectors—Device Characteristics and Data Storage Pattern Dependencies”, IEEE Trans. on Electron Devices, 2010.
- [58] L. Zhang, “compact modeling of TiO₂-TiO₂-x memristor”, APL,2013.
- [59] S. Yu, “Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory”, APL, 2011.
- [60] B. Liu, “Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine”,DAC, 2013.
- [61] L. Chua, “Memristor-the missing circuit element”, IEEE Trans. On Circuit Theory, 1971.
- [62] D. B. Strukov, et al., “The Missing Memristor Found,” Nature, 2008.
- [63] K. Kim, et al., “A Functional Hybrid Memristor Crossbar-Array/CMOS System for Data Storage and Neuromorphic Applications”, Nano Letters, 2010.
- [64] P. López, et al., “A Computational Study of the Diffuse Neighbourhoods in Biological and Artificial Neural Networks” IJCCI, 2009.
- [65] S. Vlassis, et al., “Precision Multi-Input Current Comparator and Its Application to Analog Median Filter Implementation” AICSP,2003.
- [66] N. Halko, et al.,“Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions”, Sep. 2009.
- [67] I. Tetko, D. Livingstone, and A. Luik, “Neural network studies. 1. Comparison of overfitting and overtraining,” J. Chem. Inf. Comput. Sci., pp.826-833, 1995.

- [68] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences of the USA, 1982.
- [69] S. R. Lee, et al., "Multi-level Switching of Triple-layered TaOx RRAM with Excellent Reliability for Storage Class Memory," Symposium on VLSI Technology, 2012.
- [70] X. Y. Wang, et al., "A Compact High-Accuracy Rail-to-Rail CMOS Operational Amplifier", ICBBE, 2010.
- [71] P. Dimo, "Nodal Analysis of Power Systems". Abacus Press Kent, 1975.
- [72] D. Maltoni, D. Maio, A.K. Jain and S. Prabhakar, Handbook of Fingerprint Recognition (Second Edition), Springer (London), 2009.
- [73] Miao Hu, Hai Li, Yiran Chen, Qing Wu, Garrett S Rose, "BSB training scheme implementation on memristor-based circuit," CISDA, 2013.
- [74] Golub, G. H. and Reinsch, C., "Singular value decomposition and least squares solutions," Numerische Mathematik, 1970.
- [75] P. Xie and et al, "Crypto-nets: neural networks over encrypted data", ICRL, 2015.
- [76] L. Deng, "The MNIST database of handwritten digit images for machine learning research,' IEEE Signal Processing Magazine, 2012.
- [77] Cristian Buciluă, Rich Caruana, Alexandru Niculescu-Mizil, "Model compression," international conference on Knowledge discovery and data mining, pp. 535-541, 2006.
- [78] T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction," Springer, 2009.
- [79] U. Vural, and Akgul, Y.S, "A machine learning system for human-in-the-loop video surveillance," International Conference on Pattern Recognition (ICPR), pp 1092 – 1095, 2012.
- [80] M. Barreno, B. Nelson, A.D. Joseph and J.D. Tygar, "The Security of Machine Learning," Machine Learning Journal, vol. 81, no. 2, pp. 121–148, 2010.
- [81] J. Eggermont , J. N. Kok and W. A. Kosters, "Genetic Programming for data classification: partitioning the search space," SAC, Eibe Frank and Mark Hall, Visualizing Class Probability Estimators, Practice of Knowledge Discovery in Databases, 2003.
- [82] T. Chang, S.-H. Jo and W. Lu, "Short-Term Memory to Long-Term Memory Transition in a Nanoscale Memristor," ACS Nano, pp. 7669–7676, 2011.

- [83] A. Mazady, M.T. Rahman, D. Forte and M. Anwar, “Memristor PUF—A Security Primitive: Theory and Experiment,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 5, no. 2, pp. 222-229, 2015.
- [84] J. Rajendran, G.S. Rose, R. Karri and M. Potkonjak, “Nano-PPUF: A Memristor-Based Security Primitive,” IEEE Computer Society Annual Symposium on VLSI, pp. 84-87, 2012.
- [85] F. Pedregosa, “Scikit-learn: Machine Learning in Python,” The Journal of Machine Learning Research, 2011.