In-memory computing by RRAM for machine learning

Ni, Leibin

2018

Ni, L. (2018). In-memory computing by RRAM for machine learning. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73177>

<https://doi.org/10.32657/10356/73177>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**IN-MEMORY COMPUTING BY RRAM FOR
MACHINE LEARNING**

Thesis

Submitted to School of Electrical and Electronic Engineering
of Nanyang Technological University

by

NI LEIBIN

for the Degree of Doctor of Philosophy

December 11, 2017

Acknowledgment

Firstly, I would like to express my sincerest gratitude to my advisors Prof. Yu Hao and Prof. Zhou Xing for giving me the opportunity to pursue my Ph.D. degree. I want to thank him for his countless support and valuable guidance throughout the years, without which I could not grow as a researcher. His profound knowledge has enlightened me to come up with many ideas; his scholarly inputs helped me greatly to improve many manuscripts of my works; his generosity on conference travel support allowed me to have the chances to engage with the best researchers around the world. He taught me to always set high standards for myself and made me realize the importance of global competence. For me, he sets a good example of what a researcher should be like: have enthusiasm towards research, always think out of box, and work hard to solve problems.

Secondly, I want to express my deepest appreciation to the faculty members, Prof. Anupam Chattopadhyay, Prof. Chang Chip Hong, Prof. Shi Guoyong, Prof. Huang Guangbin, Prof. J. Joshua Yang, Prof. Qiangfei Xia, who have given me a lot of guidance during my research. I also want to thank the Virtus, Chipes and EEE GSO staffs Lim-Tan Gek Eng, David Robert Neubronner, Christina Wong, and many other staffs who have been really friendly and patient to help me many times on administrative matters. Moreover, I would like to express my thanks to my wonderful colleagues and group members, Sai Manoj P. D., Song Yang, Huang Xiwei, Wang Yuhao, Xu Dongjun, Liu Xu, Jiang Yu, Liang Yuan, Yang Chang, Huang Hantao, Cai Yuehua, Xu Hang, Liu Zichuan, Liu Wenye, Suman Deb, Wang Dongzhe, Li Jinhan, Liu Mengru, Wang Zhongrui, Song Wenhao, Rao Mingyi, Zhuo Ye, Rivu Midya, and many other people who accompanied me to walk through my most memorable years.

Thirdly, I should give great appreciation to my family, my mother, my father, my grandmother, my grandfather and other relatives for their support. They have helped me a lot for concentrating on pursuing the degree, and I feel happy and lucky to have them all with me.

Finally, I would like to thank all the Singapore government sponsors for their endorsements including MOE Tier-2 (MOE2015-T2-2-013), NRF-CRP (NRF-CRP9-2011-01, NRFCRP16-2015-03), and sponsors from industries (Intel Corporation and Huawei

Technologies Corporation, Ltd). My research cannot be completed without their financial endorsements.

Abstract

The Internet data has reached exa-scale (10^{18} bytes), which has introduced emerging need to re-examine the state-of-the-art hardware architectures for data-oriented computing. There is an increasing need in the current artificial intelligence and machine learning developments. The traditional pure software-based platform cannot satisfy such an increasing need for data analytics. To analyze such a huge volume of data, we need a scalable hardware platform for both data storage and processing. The main bottleneck is from the well-known memory bottle-neck. As such, one needs to design a novel energy-efficient hardware architecture that is capable to perform the future big-data driven application.

There are critical challenges in the traditional semiconductor memory technologies. With the scaling down to nanoscale, many problems appear such as process variation, leakage current and I/O bandwidth limitations. The recent emerging resistive random-access-memory (RRAM) has shown great potential to be the solution for data-intensive applications. Besides the minimized leakage power due to non-volatility, RRAM in crossbar structure has been exploited as computational elements. As such, both memory and logic components can be realized in a power- and area- efficient manner. More importantly, it can provide a true in-memory logic-memory integration architecture without using I/Os. Therefore, the RRAM based architecture is considered as a potential universal memory for the future big-data applications.

In this PhD thesis, we will explore the development of RRAM based logic operations and NVM in-memory architecture, as well as mapping details of various machine learning algorithms on such an architecture.

Firstly, for the RRAM based in-memory computing implementation, NVM-SPICE is used for simulation for RRAM based circuits in circuit level. The measurement results of forming, SET and RESET processes are also shown. For the diffusive-type RRAM, a sub-circuit model is used for simulation. A binary RRAM-crossbar for matrix-vector multiplication is developed. Due to the non-uniformity of RRAM device, the proposed binary RRAM-crossbar can perform the computation more accurately compared to traditional RRAM-crossbar in analogue fashion. In addition, three kinds of crossbar struc-

tures including passive array, 1T1R and 1S1R are discussed. An RRAM based coupled oscillator network for L2-norm calculation is also shown. A simple oscillator can be built based on the ON-OFF switching of diffusive-RRAM or the forming process of drift-RRAM. When the basic oscillators form a network, it can perform an L2-norm calculation based on the fitted simulation results.

Secondly, for the in-memory accelerator, a distributed in-memory computing architecture (XIMA) is developed. Apart from traditional store and load, we define two more commands: start and wait. In addition, all the logic blocks and data blocks are formed in pairs so that the delay of data-processing communication can be minimized. CMOS logic such as instruction queue and decoders are also designed. In addition, 3D CMOS-RRAM architecture is investigated. For single-layer 3D architecture, RRAM devices are vias connecting top-layer wordlines and bottom-layer bitlines. All the other CMOS logics are implemented in the bottom layer. For multi-layer 3D architecture, we use two RRAM layers as data buffer and logic implementation, respectively. In addition, TSVs are used to connect different layers.

Thirdly, for the data analytics mapping on RRAM based architecture, we have accelerated three machine learning algorithms on XIMA. The learning and inference procedures of single-layer feed-forward neural network (SLFN) have been optimized and partially mapped on the passive binary RRAM-crossbar. In addition, we mapped the binary convolutional neural network (BCNN) on both passive array and 1S1R array with different mapping schemes. Moreover, L2-norm gradient based learning and inference are also implemented on an RRAM network with both crossbar and coupled oscillators. As for 3D CMOS-RRAM architecture, we also mapped the optimized SLFN algorithm. All the operations in learning and inference stages are implemented so that it can achieve the on-line learning. In addition, tensorized neural network (TNN) is mapped on both single-layer and multi-layer accelerators with different mapping scheme.

Contents

Acknowledgment	i
Abstract	iii
List of Figures	ix
List of Tables	xiii
1 Introduction	1
1.1 Motivation	1
1.1.1 Memory Wall and Power Wall	1
1.1.2 Semiconductor Memory	2
1.1.3 Non-Volatile In-Memory Computing Architecture	3
1.2 Challenges and Contributions	7
1.3 Thesis Organization	10
2 RRAM Device and Model	11
2.1 RRAM Device and SPICE Model	11
2.1.1 Drift-Type RRAM Device	11
2.1.2 Diffusive-Type RRAM Device	20
2.2 RRAM-Crossbar Structure	21
2.2.1 Analog and Digitized RRAM-Crossbar	21
2.2.2 Connection of RRAM-Crossbar	24
2.3 RRAM Based Oscillator	26
3 Machine Learning Algorithms	29
3.1 Support Vector Machine Based Machine Learning	29
3.2 Single Layer Feed-Forward Neural Network Based Machine Learning	30
3.2.1 Single Layer Feed-Forward Network	30
3.2.2 L2-Norm-Gradient Based Learning	34
3.3 Deep Convolutional Neural Network Based Machine Learning	37
3.3.1 Deep Learning for Multi-Layer Neural Network	37
3.3.2 Convolutional Neural Network	38

3.3.3	Binary Convolutional Neural Network (BCNN)	38
3.4	Tensorized Neural Network Based Machine Learning	43
3.4.1	Tensor-Train Decomposition and Compression	43
3.4.2	Tensor-Train Based Neural Network (TNN)	44
4	In-Memory Machine Learning Architecture	47
4.1	RRAM Network Based Logic Operations	47
4.1.1	RRAM-Crossbar Network	47
4.1.2	Coupled RRAM Oscillator Network	55
4.2	RRAM Network Based In-Memory Machine Learning Accelerator . . .	58
4.2.1	Distributed RRAM-Crossbar In-Memory Architecture (XIMA)	58
4.2.2	3D XIMA	61
5	Mapping Machine Learning on RRAM	65
5.1	Machine Learning Algorithms on XIMA	65
5.1.1	SLFN Based Learning and Inference Acceleration	65
5.1.2	BCNN Based Inference Acceleration on Passive Array	67
5.1.3	BCNN Based Inference Acceleration on 1S1R Array	71
5.1.4	L2-Norm Gradient Based Learning and Inference Acceleration .	73
5.2	Machine Learning Algorithms on 3D Multi-Layer CMOS-RRAM Architecture	77
5.2.1	On-Chip Design for SLFN	77
5.2.2	On-Chip Design for TNN	80
6	Results	89
6.1	Results on XIMA	89
6.1.1	Results of SLFN	89
6.1.2	Results of BCNN	95
6.2	Results on 3D CMOS-RRAM	106
6.2.1	Results of SLFN	106
6.2.2	Results of TNN	110
7	Conclusion and Future Work	121
7.1	Conclusion	121
7.2	Future Work	122
A	Publication List	125
A.1	Book	125

A.2	Book chapter	125
A.3	Journal	125
A.4	Conference	126
A.5	Workshop/Poster	128

List of Figures

1.1	Hierachy of different memories in a computer architecture.	3
1.2	In-memory computing architecture at memory cell level	4
1.3	Ideal logic-in-memory architecture	5
1.4	In-memory computing architecture at memory block level	6
2.1	The relations among the basic three elements (resistor, capacitor, inductor) and the prediction of the fourth element: RRAM. [22]	12
2.2	The diagram of TiO_2/TiO_{2-x} based RRAM cell structure.	13
2.3	Photo of fabricated RRAM array from HP.	14
2.4	Structure of RRAM and nonlinear effects for dynamic model: (a) slow-down effect at boundary, (b) exponential relation between drift velocity and electric field.	15
2.5	RRAM device simulation with Joglekar window function: I-V curve, which is consistent to the measured results reported.	15
2.6	Measurement results of voltage and current of a drift-RRAM in forming process.	16
2.7	Measurement results of hysteresis loop a drift-RRAM in SET and RESET processes.	17
2.8	Transient measurement results of resistance, voltage and current of a drift-RRAM in SET operation.	18
2.9	Transient measurement results of resistance, voltage and current of a drift-RRAM in RESET operation.	19
2.10	Characterization of drift-RRAM and diffusive-RRAM	20
2.11	Traditional analog-fashion RRAM crossbar with ADC and DAC	22
2.12	(a) Switching curve of RRAM under device variations (b) Programing inaccuracy for different RRAM target resistances	22
2.13	(a) Power consumption of analog-fashion RRAM crossbar (b) Area consumption of analog-fashion RRAM crossbar	23

2.14	Comparison of RRAM-crossbar implementation among (a) direct connection with sneak path (b) 1T1R with extra control signals (c) proposed sneak-path-free 1S1R without extra control signals; (d) comparison of analog and digital interfaces	26
2.15	(a) RRAM based oscillator schematic (b) Simulation result of oscillator (c) Discrete Fourier transform (DFT) analysis of oscillator	27
3.1	Demonstration of (a) linear SVM; (b) non-linear SVM	30
3.2	Fingerprint recognition by multiple SVMs	30
3.3	(a) Single-layer neural network (b) General flow of neural network training and testing	31
3.4	L2-norm-gradient based multi-layer neural network for machine learning	36
3.5	Diagram of simple deep learning with full-connect layers	37
3.6	Example of a layer in image classification by convolutional neural network (CNN).	38
3.7	Computation flow comparison between (a) conventional CNN; and (b) bitwise-parallelized CNN.	41
3.8	Block matrices and their representation by a 4-dimensional tensor . . .	45
3.9	Neural network weight tensorization and represented by tensor-train data format for parameter compression (from n^d to dnr^2)	46
4.1	Parallel digitizing step of RRAM-crossbar in matrix multiplication . . .	48
4.2	XOR step of RRAM-crossbar in matrix multiplication	49
4.3	Encoding step of RRAM-crossbar in matrix multiplication	50
4.4	RRAM-based inner-product operation	51
4.5	Hardware Performance Scalability under Different Reduced Dimension for (a) area; (b) delay; (c) energy (d) EDP	52
4.6	Calculation error comparison between multi-leveled and binary RRAM	54
4.7	Hardware Performance Scalability under Different Original Dimension for (a) area; (b) delay; (c) energy (d) EDP	55
4.8	(a) RRAM oscillator network schematic (b) Oscillator with single input (c) Oscillator with multiple input	56
4.9	Transient response of RRAM oscillator network output with (a) $C_{out} = 1pF$ (b) $C_{out} = 2pF$ (c) $C_{out} = 5pF$ (d) $C_{out} = 20pF$	57
4.10	Overview of distributed in-memory computing architecture on RRAM-crossbar	59
4.11	Detailed structure of control bus and communication protocol.	60

4.12	(a) 3D CMOS-RRAM accelerator architecture; (b) RRAM based matrix-vector multiplication engine.	61
4.13	(a) 3D multi-layer CMOS-RRAM accelerator architecture; (b) Incremental machine learning algorithm mapping flow on proposed accelerator	62
5.1	Detailed Mapping for digitalized matrix-vector multiplication (a) Parallel Digitizing (b) XOR 2 (c) Encoding (d) Legend of RRAM mapping .	66
5.2	RRAM-based inner-product operation	67
5.3	Digital RRAM-crossbar mapping for (a) bitwise convolution, (b) bitwise batch normalization, (c) bitwise max-pooling.	69
5.4	Mapping of bitwise CNN on sneak-path-free digital RRAM-crossbar . .	72
5.5	(a) Crossbar network for multiplication; (b) shifted additions	75
5.6	Mapping flow of machine learning on memristor network.	76
5.7	Detailed mapping for digitalized matrix-vector multiplication on RRAM-crossbar: (a) Parallel digitizing (b) XOR (c) Encoding and (d) Legend of mapping	78
5.8	Inner-product operation on RRAM-crossbar	78
5.9	Detailed pipelined and parallel mapping of CMOS-ASIC accelerator . .	79
5.10	Data control and synchronization on layer of CMOS with highly-parallel RRAM based processing elements	81
5.11	(a) RRAM based TNN accelerator (b) tree-based parallel partial result multiplication (c) RRAM based dot-product engine	83
5.12	Mapping details of highly-parallel 3D single-layer CMOS-RRAM accelerator for TNN	86
6.1	(a) Time consumption breakdown for output weight calculation (b) neural network training computation effort analysis (c) Multiplication analysis for neural network training ($N = 200, n = 128, L = 160$ and $m = 13$)	91
6.2	Training samples and prediction value $\mathbf{Y}$ from Equation (3.7) for face recognitions.	92
6.3	Time consumption proportion of Iris dataset (a) without RRAM network (b) with RRAM network; and energy consumption proportion of Mushroom dataset (c) without RRAM network (d) with RRAM network . . .	94
6.4	RRAM configuration with voltage amplitude variation following Gaussian distribution.	96
6.5	(a) Accuracy comparison with analog RRAM under device variation (b) accuracy comparison with truncated CNN with approximation.	97

6.6	Scalability study of testing error rate under different energy consumption with (a) layer 2 output maps, and (b) layer 3 hidden nodes; (c) comparison with layer 2 output maps and layer 3 hidden nodes.	98
6.7	Image recognition example of CIFAR-10 dataset	100
6.8	Power consumption of bitwise CNN (a) for different operations (b) for different layers.	100
6.9	Performance with different number of CONV-4 output maps in CIFAR-10.	101
6.10	Sneak-path-free effect in 1S1R implementation for write operation. . . .	102
6.11	Sneak-path-free effect in 1S1R implementation for read operation. . . .	103
6.12	Programing inaccuracy for different target resistances with voltage amplitude variation following Gaussian distribution	104
6.13	Scalability study of hardware performance with different hidden node numbers for: (a) area; (b) delay; (c) energy and (d) energy-delay-product (EDP)	106
6.14	Energy-saving comparison under different bit-width of CMOS and RRAM with the same accuracy requirement.	108
6.15	On-line machine-learning for image recognition on the proposed 3D multi-layer CMOS-RRAM accelerator using benchmark CIFAR-10 . . .	110
6.16	Scalability study of hardware performance with different hidden node numbers for: (a) area; (b) delay; (c) energy and (d) energy-delay-product	112
6.17	Testing time and accuracy comparison between tensorized neural network (TTN) and general neural network (Gen.) with varying number of hidden nodes.	113
6.18	Tensorized neural network layer 1 weight and layer 2 weight histogram with approximate Gaussian distribution.	117
6.19	Compression rate and accuracy with increasing bit-width.	118

List of Tables

2.1	Definitions of variables used for drift-RRAM device	11
4.1	Performance comparison of matrix-vector multipliation under among software and hardware implementations.	53
4.2	Detailed evaluation of RRAM based oscillator	57
4.3	Protocols between external processor and control bus.	60
5.1	Binary format stored in RRAM-crossbar	70
5.2	Bitwise CNN configuration of CIFAR-10 dataset.	71
5.3	Conversion from signed bitwise convolution to unsigned bitwise XNOR	72
6.1	Face recognition performance comparison under among software and hardware implementations.	90
6.2	Time and energy comparison on UCI dataset.	93
6.3	Bitwise CNN configuration of MNIST dataset.	97
6.4	Performance comparison among different software and hardware imple- mentations on CIFAR-10 BCNN.	102
6.5	Performance comparison under different software and hardware imple- mentations	105
6.6	Testing accuracy of machine learning techniques under different dataset and configurations (normalized to all 32 bits)	107
6.7	Performance comparison under different software and hardware imple- mentations.	109
6.8	Bandwidth improvement under different number of hidden nodes for MNIST dataset.	112
6.9	Model Compression under different number of hidden nodes and tensor ranks on MNIST dataset.	113
6.10	Testing accuracy of learning techniques under different dataset and bit- width configuration.	114

6.11	Performance comparison under different hardware implementations on MNIST dataset with 10,000 testing images.	115
6.12	Model Compression under different number of hidden nodes and tensor ranks on MNIST dataset.	117
6.13	Bandwidth improvement under different number of hidden nodes for MNIST dataset.	118
6.14	Performance comparison under different hardware implementations on MNIST dataset (10K test images).	119

Chapter 1

Introduction

1.1 Motivation

1.1.1 Memory Wall and Power Wall

Information technology by new social media such as Google and Facebook is generating a wealth of data at exa-level (10^{18} bytes) that is previously unimaginable. In-depth data analytics may become the driving force for the future economic growth in a digital data oriented society. The main challenge stems from the existing hardware infrastructure built upon the conventional computer architecture and CMOS technology. Moores laws scaling has become slower when compared to the growth of data. There are many recent large-scale research programs such as Exa-scale Computing (NSF/DARPA-USA) and Human Brain (European Commission), which are, however, still limited at this singularity point because of the memory bottleneck. During big-data analytics, it needs to process the stored data with intensive memory access. However, the state-of-the-art data storage and processing hardware still have a bandwidth-wall problem with large I/O congestion; but also power-wall with large static power in advanced CMOS technology.

In big-data-driven applications, we are looking for some implementations with high-bandwidth and low-power density. For instance, intensive memory access may occur in web searching, data mining and also machine learning applications. The microprocessor needs to process the stored data with the intensive memory access. However, state-of-the-arts data storage and processing hardware still have bandwidth-wall due to limited accessing bandwidth at I/Os; but also power-wall due to the significant leakage power in CMOS circuit when holding data. As a result, a design of scalable energy-efficient big-data analytic hardware is highly required. There are following research activities [1, 2] attempted to alleviate the memory bottleneck problem for big-data storage and processing in the future:

- Firstly, as a promising technology for big-data oriented computing, the emerging non-volatile in-memory computing shows a great potential as a high-throughput, energy-efficient solution. The application specific accelerators can be developed within memory. In this case, before read-out by processing unit, data can be pre-processed so that data migration is minimized and the bandwidth-wall can be relieved. Moreover, the non-volatile memory devices hold information without using charge such that leakage current can be significantly reduced with the power-wall relieved as well.
- Secondly, sparse-represented data by compressive sensing is an effective approach to reduce data size by projecting data from high-dimensional space to low-dimensional subspace with essential feature preserved. Data stored in memory after compressive sensing can be recovered or directly processed. This procedure reduces data complexity in data storage as well as in data analytics with a further significant power reduction and bandwidth improvement.
- Thirdly, data analytics by machine-learning accelerator can be developed for fast data analytics. Latest machine learning algorithm on-chip can accelerate the learning process with a potential to realize on-line training, which is important for applications in autonomous vehicles and unmanned aerial vehicles.

1.1.2 Semiconductor Memory

1.1.2.1 Overview

Traditional semiconductor memories refer to the silicon based transistor devices (CMOS, BTJ, etc.) in computer architectures. These memories are applied to store computation instructions as well as data temporarily or permanently. Compared to storage media such as hard disk drive (HDD) and optical disc drive (ODD) with predetermined order of data access due to the limitations of mechanical drive, the semiconductor memories possess the property of random access. It takes almost an identical time to access any data regardless of the data location.

We can further classify all the semiconductor memories into two classes: volatile memory and non-volatile memory. In volatile memory, data is stored as electrical signals such as voltage, and it requires power maintained. Once the power supply stops, the device is turned off and data is lost. Currently, the most commonly used volatile memories are static random-access memory (SRAM) and dynamic random-access memory (DRAM), whose data is indicated by the electrical voltage levels. On the contrary, non-volatile memory is able to retain the data even when the device is turned off, as its data

Figure 1.1: Hierarchy of different memories in a computer architecture.

is mostly preserved by non-electrical states. For example, bits in programmable read-only memory (PROM) are denoted by whether the fuses of individual memory cells are burned. Figure 1.1 briefly introduces the characteristics of the volatile SRAM/DRAM, traditional NVMs and also emerging NVMs.

1.1.2.2 Nano-Scale Limitations

With scaling down of the technology in nano-level, there are some challenges and limitations in the traditional memory architecture. Such challenges are caused by physical rules, and we can summarize them in two perspectives. On one hand, there are function failures due to the process variation that transistors may have mismatch problems. On the other hand, the thermal runaway failures are caused by positive feedback in an SRAM or DRAM cell. As a result, non-volatile memory should be considered as a potential solution for data storage with the scaling technology.

1.1.3 Non-Volatile In-Memory Computing Architecture

In the conventional computing architecture, all the data is stored in memory. It is separated from the general processors so that I/O connections are required. As a result, during the data processing, all data needs to be read out to the external processor, and written back afterwards. However, in data-oriented applications such as machine learning acceleration, this architecture will incur significant I/O congestions and hence greatly

Figure 1.2: In-memory computing architecture at memory cell level

degrade the overall performance [3]. In addition, significant static power will be consumed because all the data needs to be held, even though it does not need to be used at the moment.

Theoretically, it is feasible to overcome the bandwidth issue by adding more I/O pins or operating them at higher frequency. Practically, however, the I/O frequency is limited by the signal propagation delay and signal integrity issues, and I/O number is limited by the packaging technology, thus the bandwidth can hardly get further improved.

Besides improving the bandwidth of memory, reducing the volume of data communicating between processor and memory is also feasible. Conventionally, the processor only reads raw data from main memory. If the memory can perform some operations

Figure 1.3: Ideal logic-in-memory architecture

before sending data, the I/O communication requirement can be significantly reduced. For instance, if we need to add eight numbers, we need to load all the eight values into the processor in the conventional way. However if we can perform the addition of two numbers as an in-memory logic operation, we can pre-process the adding inside memory and only four numbers needs to be read out.

To perform in-memory computing, we need to implement logic operations inside memory so that pre-processing can be done. Such an architecture is called logic-in-memory architecture. In general, the requirement of the ideal logic-in-memory architecture can be summarized in Figure 1.3. A big non-volatile memory sea is connect with thousands of small accelerator cores through high Bandwidth and energy-efficient reconfigurable I/Os. Considering the leakage reduction at the same time, logic-in-memory architectures that are associated with non-volatile memory are presented in [4, 5, 6, 7, 8]. Figure 1.2 shows a logic-in-memory architecture by building CMOS circuits with storage devices. The example shows a full-adder with both *sum* and *carry* logic. However, such a cell-level in-memory architecture has two major problems. Firstly, the CMOS based logic is built in memory so that it is difficult to be reconfigured. Second, it can only perform some simple logic. In the example it needs 22 transistors for a simple 1-bit adder. The complexity of memory will be greatly increase in this solution.

Another in-memory architecture we are going to present at block level in distributed fashion is illustrated in Figure 1.4, which is more effective for traffic reduction. A mem-

Figure 1.4: In-memory computing architecture at memory block level

ory data is usually organized in H-tree fashion, and the data block can be the data array or a number of data arrays that belong to same ‘H-tree’ branch. Instead of inserting in-memory logic at memory cell level inside the data array, the architecture in Figure 1.4 pairs each block of data with in-memory logic (accelerators). Different from the cell level in-memory architecture, the accelerators can be made with higher complexity, and the number of accelerators for each data block can also be customized. The data flow of the block level in-memory architecture is to readout data from data block to in-memory logic, which performs particular functionality and then writes back the result. The data also needs to be stored in assigned blocks but it is much more flexible than that of cell level in-memory architecture. The block level in-memory architecture is very effective to reduce communication traffic between memory and processor. This is because significant operands reduction can be achieved by deploying accelerator with high level functionality. For example, for face recognition in image processing application, instead of transmitting a whole image to obtain a Boolean result, the result can be directly gained through in-memory logic. In other words, the block level in-memory architecture is suitable for big data driven applications where traffic reduction is more important than latency reduction.

In this architecture, the non-volatile resistive random-access memory (RRAM) is intensively used. Both the memory block and logic block in each pair are purely implemented by RRAM devices. In addition, energy efficient in-memory logic units are deployed in the external processor to execute instructions that cannot be accelerated by in-memory logic.

1.2 Challenges and Contributions

Future cyber-physical system requires efficient real-time data analytics [9, 10, 11, 12, 13] with applications in robotics, brain-computer interface as well as autonomous vehicles. The recent works in [14, 15] have shown a great potential for machine learning with significant reduced training time for real-time data analytics.

Hardware-based accelerator is currently practiced to assist machine learning. In traditional hardware accelerator, there is intensive data migration between memory and logic [16, 17] causing both bandwidth and power walls. Therefore, for data-oriented computation, it is beneficial to place logic accelerators as close as possible to the memory to alleviate the I/O communication overhead [18]. The cell-level in-memory computing is proposed in [19], where simple logic circuits are embedded among memory arrays. Nevertheless, the according in-memory logic that is equipped in memory cannot be made for complex logic function, and also the utilization efficiency is low as logic cannot be shared among memory cells. In addition, there is significant memory leakage power in CMOS based technology.

Emerging resistive random-access-memory (RRAM) [20, 21, 22, 23, 24, 25, 26] has shown great potential to be the solution for data-intensive applications. Besides the minimized leakage power due to non-volatility, RRAM in crossbar structure has been exploited as computational elements [21, 27]. As such, both memory and logic components can be realized in a power- and area- efficient manner. More importantly, it can provide a true in-memory logic-memory integration architecture without using I/Os. Nevertheless, the previous RRAM-crossbar based computation is mainly based on an analog fashion with multi-level values [28] or Spike Timing Dependent Plasticity (STDP) [29]. Though it improves computation capacity, the serious non-uniformity of RRAM-crossbar at nano-scale limits its wide applications for accurate and repeated data analytics. Moreover, there is significant power consumption from additional AD-conversion and I/Os mentioned in [29]. Besides RRAM-crossbar, coupled RRAM-oscillator can be utilized for analog computing. By mapping fixed-point values to voltages, coupled RRAM-oscillator can perform as L2-norm calculation. Significant speed-up and energy-efficiency can be achieved due to the direct analog computing.

Although the non-volatile in-memory computing seems to be a promising solution for future data-oriented storage and computing system, the recent emerging RRAM are still in their infancy due to the following unresolved challenges:

- From circuit level, although there exists some simulators to perform hybrid NVM-CMOS circuit simulation [25, 26], the RRAM based computation operations have not been fully developed yet. Without the detailed in-memory computing implementation on RRAM, it is hard to perform a real application on such an RRAM based non-volatile in-memory accelerator.
- From architecture level, it is necessary to explore a detailed implementation for the in-memory architecture. Controllers for conventional memory are not optimal for the RRAM based in-memory computing system. The CMOS based control bus and its communication protocol need to be re-defined for such a novel architecture.
- From system level, it is necessary to find out what machine learning algorithms can be mapped on the in-memory accelerator. For different operations or algorithms, we need to design different accelerators in order to achieve a higher parallelism and a better energy-efficiency.

This PhD thesis has explored the development of RRAM based logic operations and NVM in-memory architecture, as well as mapping details of various machine learning algorithms on such an architecture. The main contributions of this thesis are listed as follows.

For the RRAM based in-memory computing implementation, the target is to develop detailed mappings for different operations on RRAM. We proposed two kinds to RRAM based structures for operation accelerating: RRAM-crossbar and RRAM oscillator network. Particularly, this PhD thesis has summarized the following works:

- Firstly, the NVM-SPICE is used for circuit level simulation. The model of drift-type RRAM is introduced with simulation results. In addition, the measurement results of forming, SET and RESET processes are also shown. For the diffusive-type RRAM, a sub-circuit model is used for simulation.
- Secondly, we developed a binary RRAM-crossbar for matrix-vector multiplication. Due to the non-uniformity of RRAM device, the proposed binary RRAM-crossbar can perform the computation more accurate compared to traditional RRAM-crossbar in analogue fashion. In addition, three kinds of crossbar structures including passive array, 1T1R and 1S1R are discussed.

- Thirdly, we developed an RRAM based coupled oscillator network for L2-norm calculation. A simple oscillator can be built based on the ON-OFF switching of diffusive-RRAM or the forming process of drift-RRAM. When the basic oscillators form a network, it can perform an L2-norm calculation based on the fitted simulation results.

For the in-memory accelerator, the target is to determine the design of architecture. For the architecture with only RRAM as computing engine, we design the auxiliary CMOS logic and define the communication protocol between the processor and memory in detail. For the architecture with both RRAM and CMOS as computing engine, we develop 3D CMOS-RRAM architecture so that the a higher throughput and parallelism can be achieved. Particularly, this PhD thesis has summarized the following works:

- Firstly, a distributed in-memory computing architecture (XIMA) is developed. Despite of traditional store and load, we define two more commands: start and wait. In addition, all the logic blocks and data blocks are formed in pairs so that the delay of data-processing communication can be minimized. CMOS logic such as instruction queue and decoders are also designed.
- Secondly, 3D CMOS-RRAM architecture is investigated. For single-layer 3D architecture, RRAM devices are vias connecting top-layer wordlines and bottom-layer bitlines. All the other CMOS logics are implemented in the bottom layer. For multi-layer 3D architecture, we use two RRAM layers as data buffer and logic implementation, respectively. In addition, TSVs are used to connect different layers.

For the data analytics mapping on RRAM based architecture, the target is to focus on detailed accelerator design for specific applications. In addition, the algorithm requires to be optimized so that all the operations can be fully mapped on the accelerator. We have mapped several machine learning algorithms on XIMA and 3D CMOS-RRAM architectures with significant improvement on bandwidth and energy-efficiency. Particularly, this PhD thesis has summarized the following works:

- Firstly, for the distributed in-memory architecture (XIMA), we have accelerated three machine learning algorithms. The learning and inference procedures of single-layer feed-forward neural network (SLFN) have been optimized and partially mapped on the passive binary RRAM-crossbar. In addition, we mapped the binary convolutional neural network (BCNN) on both passive array and 1S1R array with different mapping schemes. Moreover, L2-norm gradient based learning

and inference are also implemented on an RRAM network with both crossbar and coupled oscillators.

- Secondly, for 3D CMOS-RRAM architecture, we also mapped the optimized SLFN algorithm. All the operations in learning and inference stages are implemented so that it can achieve the on-line learning. In addition, tensorized neural network (TNN) is mapped on both single-layer and multi-layer accelerators with different mapping scheme.

For every machine learning algorithm mapped on XIMA and 3D CMOS-RRAM accelerator, we evaluate their performance in device, architecture and system levels. All the implementations show higher throughput, bandwidth and parallelism with better energy-efficiency.

1.3 Thesis Organization

This thesis covers the entire flow from device, architecture and learning algorithm to system perspectives for emerging RRAM. It is organized into following chapters. Chapter 2 introduces the characterization of RRAM devices and how to utilize them as computation components. The structure of digitalized RRAM-crossbar and RRAM based oscillator are shown. Chapter 3 introduces different machine learning algorithms including SVM and neural network based learning. Various types of the neural networks are compared in detail. Chapter 4 proposes two types of RRAM-crossbar based accelerators. Distributed in-memory architecture and 3D CMOS-RRAM architecture with single-layer and multi-layer are introduced. In addition, it details the mapping of RRAM network for computing, including RRAM-crossbar and RRAM oscillator. Chapter 5 shows the detailed mapping of machine learning algorithms on various RRAM based computing architectures. Chapter 6 shows the performance of RRAM-based accelerators for machine learning algorithms, and Chapter 7 summarizes the thesis and shows some potential topic as future work.

Chapter 2

RRAM Device and Model

In this chapter, we will firstly introduce the characteristics of RRAM devices. After that, the basic structure consisting of RRAMs will be discussed.

2.1 RRAM Device and SPICE Model

In this section, we will illustrate two types of the RRAM devices: drift-type and diffusive type. The model and usage of these devices are introduced in [30, 2], and we will show them in detail.

2.1.1 Drift-Type RRAM Device

Table 2.1: Definitions of variables used for drift-RRAM device

Variables	Definitions
M	resistance of RRAM
G	conductance of RRAM
Φ	magnetic flux
q	charge
W	Length of the doped region
D	thickness of RRAM film
$w(t)$	doping ratio (W/D), $w(t) \in [0, 1)$ (non-volatile state variable)
$dw(t)/dt$	normalized drift velocity
R_{LRS}	resistance of low-resistance-state (LRS)
R_{HRS}	resistance of high-resistance-state (HRS)
$F(x)$	window function defining the boundary condition of $w(t)$
μ	mobility at small electric field
E_0	characteristic field for a particular mobile atom in the crystal

Figure 2.1: The relations among the basic three elements (resistor, capacitor, inductor) and the prediction of the fourth element: RRAM. [22]

The existence of resistive random-access memory (RRAM), also named as memristor was first predicted by Leon Chua in 1976 [22], but not until nearly 30 years later was it first discovered in nano-scale devices in HP Labs [24]. The name memristor is derived from the fact that its resistance is determined by the current passed through in the past as if it can remember its history. The memory effect specifically is the time integral of the current flowing through the device: when current flows in one direction through an RRAM, its electrical resistance increases; when current flows in the opposite direction, resistance decreases; and when the current is stopped, the RRAM retains its previous resistance. The bistable states of the device are defined as high resistance state and low resistance state. The write operation is achieved by applying large current, which changes its resistance rapidly. To read the cell, small current is applied to detect its resistance without significantly changing its resistance. The foundation of the forth element RRAM proposed by Leon Chua is shown in Figure 2.1. In 2008, the first physical realization of an RRAM was demonstrated by HP Labs: the memristive effect was achieved by moving the doping front along a TiO_2 thin-film device [24]. The materials at the different sides of the doping front have different resistivity, and the overall resistance is calculated as the two resistors in series. The diagram of $\text{TiO}_2/\text{TiO}_{2-x}$ based RRAM cell structure is shown in Figure 2.2. In addition, HP Labs also shows the STM photo of RRAM array (the RRAM crossbar structure) in [23], as shown in Figure 2.3.

A comprehensive dynamic RRAM model which takes charge-induced-drifting ef-

Figure 2.2: The diagram of TiO_2/TiO_{2-x} based RRAM cell structure.

fect, slow-down effect and strong-electric-field effect into consideration is illustrated in Figure 2.4(a) with doping and un-doping regions. The term doping ratio (w) is introduced to numerically represent the ratio of doping region (W) over the whole thickness of RRAM film (D). As there is a large difference of resistivity in doping and un-doping regions, the total resistance is changing with doping ratio. For instance, when $w = 1$ the RRAM is at low-resistance-state (LRS), where the resistance is a thousand times smaller than the resistance at high-resistance-state (HRS) with $w = 0$.

The operating principle of RRAM device can be summarized as: charge-induced state (doping-ratio w) shifting. As illustrated in Figure 2.4(a), the RRAM can be seen as two regions in series, doping region and undoping region. The domain-wall that separates two regions can be shifted by electric charge that flows through. When the positive charge flows towards the right hand side of RRAM junction, the domain-wall shifts rightward thus the doping ratio (w) increases and the resistance reduces accordingly, and vice-verse for the charge flowing in the reverse direction. Traditionally, this mechanism can be described as generalized voltage-controlled memristive system [31] with the following equations:

$$v(t) = R(w, v) i(t) \quad (2.1)$$

$$\frac{dw(t)}{dt} = f(w, v) \quad (2.2)$$

where R is the resistance of RRAM and f is the explicit function of w and applied voltage v ; dw/dt is the normalized drift velocity determined by w and external voltage v . Note that the complementary current-controlled memristive system can be defined

Figure 2.3: Photo of fabricated RRAM array from HP.

similarly. The drifting velocity of RRAM is a function of applied current that can be initially modeled in the following piecewise linear form [24]:

$$\frac{dw(t)}{dt} = \begin{cases} \mu \frac{R_{LRS}}{D^2} i(t) & \text{mode1 : normal operation} \\ 0 & \text{mode2 : } w = 0 \text{ or } 1 \\ 0 & \text{mode3 : } |v| < V_{th} \end{cases} \quad (2.3)$$

In addition, strong-electric-field effect stands for the exponential increase in the drift velocity under a strong electrical field condition ($> 1MV/cm$) [32]. It usually happens in the nano-scale devices such as newly fabricated RRAM at scale of $20nm$. As illustrated in Figure 2.4(b), the phenomenon can be approximated by a sinh function [32]:

$$\frac{dw(t)}{dt} = \frac{\mu E_0}{D} \sinh\left(\frac{E}{E_0}\right) \quad (2.4)$$

where $E = v(t)/D$ is the applied field strength, and definitions of remaining variables are shown in Table 2.1.

Moreover, slow-down effect of RRAM is illustrated in Figure 2.4(a), when w is ap-

Figure 2.4: Structure of RRAM and nonlinear effects for dynamic model: (a) slow-down effect at boundary, (b) exponential relation between drift velocity and electric field.

proaching the boundaries where $w = 0$ or 1 , the drift velocity drops quickly and is no longer linear with the current applied. This nonlinear dynamic effect can be characterized by either of the following window functions [33, 34]:

$$F(w) = 1 - (2w - 1)^{2p} \quad (2.5)$$

Figure 2.5: RRAM device simulation with Joglekar window function: I-V curve, which is consistent to the measured results reported.

Figure 2.6: Measurement results of voltage and current of a drift-RRAM in forming process.

$$F(w) = 1 - [(w - u(-i(t)))^{2p}] \quad (2.6)$$

Therefore, the complete nonlinear dynamic RRAM model relevant to doping-factor w considering charge-induced-drifting effect, slow-down effect and strong-electric-field effect can be obtained from Equations (2.2, 2.3, 2.4, 2.5, 2.6):

$$\begin{aligned} \frac{v(t)}{i(t)} &= R_{HRS} - (R_{HRS} - R_{LRS})w(t), \\ \frac{dw(t)}{dt} &= \frac{\mu E_0}{D} \sinh\left(\frac{v(t)}{DE_0}\right) F(w(t)) \end{aligned} \quad (2.7)$$

Applying Equations (2.5), (2.6) and (2.7) derived in last section to the RRAM dynamic function described in MNA matrix, we can obtain the required Jacobian terms as follows

$$\begin{aligned} G &= \begin{bmatrix} G & -G \\ -G & G \end{bmatrix}; S = \begin{bmatrix} v_n \times \frac{dG}{dw_m} \\ -v_n \times \frac{dG}{dw_m} \end{bmatrix} \\ K_v^F &= \begin{bmatrix} -c_1 c_2 F(w_m) \cosh(c_2 v_n) & c_1 c_2 F(w_m) \cosh(c_2 v_n) \end{bmatrix}; \end{aligned}$$

Figure 2.7: Measurement results of hysteresis loop a drift-RRAM in SET and RESET processes.

$$K_s^F = -c_1 \sinh(c_2 v_n) \times \frac{dF(w_m)}{dw_m}; K_v^G = 0; K_s^G = 1,$$

where G is the conductance and $\frac{dG}{dw_m}$ is the non-volatile state variable derivative of conductance; $c_1 = \frac{\mu E_0}{D}$, $c_2 = \frac{1}{DE_0}$ are constants; and $F(w)$ is the window function. Recall that the other parameters are defined in Table 2.1. Therefore, all Jacobian terms required for the linearized system equation can be established, with which the RRAM device simulation considering the non-volatile state variables can be implemented in the SPICE-like simulator accordingly.

In order to verify the dynamic RRAM model and our simulator, simulation to generate RRAM hysteresis loop has been conducted to validate against experimental measurement in [24]. The device parameters are set as $d = 10\text{nm}$, $R_{LRS} = 200\Omega$, $R_{HRS} = 3800\Omega$; the initial resistance $R_{init} = 3790\Omega$, $\mu = 1 \times 10^{-17} \text{m}^2/(\text{V} \cdot \text{s})$, $E_0 = 1\text{MV}/\text{cm}$; and Joglekar window function is applied with $p = 2$. The dynamic behavior is investigated under the driving voltage of $v(t) = -0.2 + \sin(200\pi t)$. As shown in Figure 2.5, simulation result can well capture the experiment results reported in [24].

Figure 2.8: Transient measurement results of resistance, voltage and current of a drift-RRAM in SET operation.

Apart from the modeling and simulation results, we also have some measurements of the drift-RRAM device. We need to perform a forming process of the RRAM first, as shown in Figure 2.6. Compliance current is set to $50\mu A$ in our experiment.

After forming process, we also do some experiments for the SET and RESET of the RRAM device. Firstly we have a DC measurement with 17 times of SET and 18 times of RESET, and they forms a hysteresis loop as shown in Figure 2.7. For the SET process,

Figure 2.9: Transient measurement results of resistance, voltage and current of a drift-RRAM in RESET operation.

we use $10\mu A$ as the compliance current.

We also perform some transient measurements. Three pulses are used that the first and third pulse is for read propose, and the second voltage pulse is for SET or RESET. The transient measurement results use $1\mu s$ as time interval. Firstly, we use Figure 2.8 to show the SET operation, and the resistance of RRAM changes from $1M\Omega$ to $50k\Omega$. Here, a $0.5V$ pulse is for read operation while $2.5V$ is for write. We can see the SET delay is in $\sim\mu s$ so that we can hardly see the procedure of resistance change. Secondly, we use Figure 2.9 to show the RESET operation, and the resistance of RRAM changes from 680Ω to $100k\Omega$. A $0.5V$ pulse is for read operation while $-2V$ is for write.

Figure 2.10: Characterization of drift-RRAM and diffusive-RRAM

2.1.2 Diffusive-Type RRAM Device

The recent research studies [35, 36] have reported a diffusive-type RRAM device. Different from the common drift-type RRAM, the initial state of diffusive RRAMs is always HRS with $\sim G\Omega$ when no external voltage or current is applied. When the input voltage exceeds a threshold value $V_{th,on}$ is applied for a while (e.g. 1ms), the diffusive RRAM will switch to LRS with $\sim k\Omega$. After the device switches on, the resistance will slowly recover back to HRS if the applied voltage drops below another threshold $V_{th,off}$ for about 20ms. Such a device shows great potentials as an automatically ON/OFF selector. In the followings, we use R_{on} and R_{off} to represent the two states of diffusive-type RRAM.

In mosr application, the diffusive RRAM is conneted to a drift RRAM serially, forming a one-selector-one-RRAM (1S1R) structure. Such a diffusive RRAM can work as the synapse emulator [35], and also the selectors [36]. We use SPICE simulation to characterize both of the RRAMs in Figure 2.10. Here we set $R_{LRS} = 500k\Omega$, $R_{HRS} = 5M\Omega$ according to [37] for drift-type RRAM, and $R_{on} = 9k\Omega$, $R_{off} = 90M\Omega$ similar to [36] for diffusive-type RRAM. In the following sections, the SPICE model is used for 1S1R performance evaluation.

2.2 RRAM-Crossbar Structure

As mentioned above, RRAM is a two-terminal device that can be observed in sub-stoichiometric transition metal oxides (TMO_s) sandwiched between metal electrodes. Such a device can be used as non-volatile memory with state of ion resistance, which results in 2 non-volatile states: high resistance state HRS and low resistance state LRS. One can change the state from HRS to LRS or vice versa by applying a SET voltage (V_w) or a RESET voltage ($-V_w$).

The two states HRS and LRS represent 0 and 1, respectively. To read a RRAM cell, one can apply a read voltage V_r to the RRAM. The V_r and V_w follow

$$V_w > V_{th} > V_w/2 > V_r, \quad (2.8)$$

where V_{th} is the threshold voltage of the RRAM.

Because of the high density of RRAM device, one can build a crossbar structure as the array of RRAM [21, 38, 39, 40, 41, 42]. Such crossbar structure can be utilized as memory for high-density data storage. The memory array can be read or written by controlling the voltage of wordlines (WLs) and bitlines (BLs). For example, we can apply $V_w/2$ on the i^{th} WL and $-V_w/2$ on the j^{th} BL to write data into the RRAM cell on i^{th} row, j^{th} column.

2.2.1 Analog and Digitized RRAM-Crossbar

We will introduce the traditional analog RRAM-crossbar and the digitized RRAM-crossbar in the followings. The I/O interface and the number of RRAM device resistance value is different in these two structures.

2.2.1.1 Traditional Analog RRAM-Crossbar

RRAM is an emerging non-volatile memory based on two-terminal junction devices, whose resistance can be controlled by the integral of externally applied currents. The fabric of crossbar intrinsically supports matrix-vector multiplication where vector is represented by row input voltage levels and matrix is denoted by mesh of RRAM resistances. Suppose we want to calculate the matrix-vector multiplication $y = \Phi x$. As shown in Figure 2.11, by configuring Φ into the RRAM crossbar and setting x as the crossbar input, the analog computation can be achieved by RRAM crossbar with directly output y .

However, such analog RRAM-crossbar has two major drawbacks. Firstly, the pro-

Figure 2.11: Traditional analog-fashion RRAM crossbar with ADC and DAC

gramming of continuous-valued RRAM resistance is practically challenging due to large RRAM process variation. Specifically, the RRAM resistance is determined by the integral of current flowing through, which leads to a switching curve as shown in Figure 2.12 (a). With the process variation, the curve may shift and leave intermediate values very unreliable to program, as shown in Figure 2.12 (b). Secondly, the A/D and D/A converters are both timing-consuming and power-consuming. In our simulation, the A/D and D/A conversion may consume up to 85.5% of total operation energy in 65nm as shown in Figure 2.13.

2.2.1.2 Proposed Digitalized RRAM-Crossbar

To overcome the aforementioned issues, we propose a full-digitalized RRAM-crossbar for matrix-vector multiplication. Firstly, as ON-state and OFF-state are much more

Figure 2.12: (a) Switching curve of RRAM under device variations (b) Programming inaccuracy for different RRAM target resistances

reliable than intermediate values shown in Figure 2.12, only binary values of RRAM are allowed to reduce the inaccuracy of RRAM programming. Secondly, we deploy a pure digital interface without A/D conversion.

In RRAM crossbar, We use V_{wl}^i and V_{bl}^j to denote voltage on i th wordline (WL) and j th bitline (BL). R_{LRS} and R_{HRS} denote the resistance of low resistance state (LRS) and high resistance state (HRS), respectively. In each sense amplifier (SA), there is a sense resistor R_s with fixed and small resistance. The relation among these three resistance is $R_{HRS} \gg R_{LRS} \gg R_s$. Thus, the voltage on j th BL can be presented by

$$V_{bl}^j = \sum_{i=1}^m g_{ij} V_{wl}^i R_s \quad (2.9)$$

where g_{ij} is the conductance of R_{ij} .

The key idea behind digitalized crossbar is the use of comparators. As each column output voltage for analog crossbar is continuous-valued, comparators are used to digitize it according to the reference threshold applied to SA in Figure 4.10,

$$O_j = \begin{cases} 1, & \text{if } V_{bl}^j \geq V_{th}^j \\ 0, & \text{if } V_{bl}^j < V_{th}^j \end{cases} \quad (2.10)$$

However, the issue that rises due to the digitalization of analog voltage value is the loss of information. To overcome this, three techniques are applied. Firstly, multi-thresholds are used to increase the quantization level so that more information can be preserved. Secondly, the multiplication operation is decomposed into three sub-operations that binary crossbar can well tackle. Thirdly, the thresholds are delicately

Figure 2.13: (a) Power consumption of analog-fashion RRAM crossbar (b) Area consumption of analog-fashion RRAM crossbar

selected at the region that most information can be preserved after the digitalization.

Compared to the traditional analog RRAM-crossbar [43, 44], we can summarize the advantages of the digital RRAM-crossbar in the following perspectives:

- It has better programming accuracy of the RRAM device under process variation with no additional ADC conversation as well.
- Since only LRS and HRS are configured in digital RRAM-crossbar, it does not require high HRS/LRS ratio so that low-power (high LRS) device [37] can be applied. In addition, the wire resistance affects little on IR-drop when using the low-power RRAM device.
- The binary input voltage has a better robustness on the IR-drop in large-size crossbar.

However, the area and energy consumption of RRAM devices will increase in the digital RRAM-crossbar implementation. It will also be discussed in Section 4.

2.2.2 Connection of RRAM-Crossbar

There are three approaches for the connection of RRAM-crossbar: direct-connected RRAM, one-transistor-one-RRAM (1T1R), and one-selector-one-RRAM (1S1R). We will introduce all these approaches and specify the differences among them.

2.2.2.1 Direct-Connected RRAM

Direct-connected RRAM is also called the passive RRAM array, which is a crossbar structure with only RRAM devices. Every WL and BL is connected with a single drift-type RRAM. The structure of direct-connected RRAM is shown in Figure 2.14(a), and here we use i^{th} WL (WL_i) and j^{th} BL (BL_j) as an example. To perform a write operation, one needs to apply a write voltage V_{write} between WL_i and BL_j . In this case, the RRAM connecting them ($R_{i,j}$) can be precisely configured.

However, a problem exists in this structure that sneak-path may occur. In this example, the purple path is the ideal current path we want to have, but there may be another path identified by green. This path consists of $R_{i-1,j}$, $R_{i-1,j-1}$ and $R_{i,j-1}$ so that we can see a V_{write} is also applied on the serial of these three RRAMs. In some of the cases, two of them are in LRS and one of them is in HRS, causing a very large voltage applied on the RRAM device in HRS. In this case, the status of this RRAM may also be configured, which we do not want it happens.

To overcome the sneak-path issue, a solution is proposed in [45, 46, 47] that we can applied a constant voltage on the rest of WLs and BLs. Here, the target WL and BL are applied with $2/V_{write}$ and $-2/V_{write}$, respectively. Meanwhile, we also apply 0V on all the inactive lines. This solution has two requirements for the device and controller design:

- The non-linearity of RRAM device should be very good needs since the device should not be configured with a $2/V_{write}$ drop.
- The external CMOS controller needs to output multiple voltages for all lines at the same time. In addition, the amplitude of voltage of all the lines should be independent.

2.2.2.2 One-Transistor-One-RRAM (1T1R)

In one-transistor-one-RRAM (1T1R) structure, an NMOS transistor is connected with each RRAM device serially indicated in [43]. As a result, each WL is connected to a BL with one transistor and one RRAM, as shown in Figure 2.14(b). The NMOS transistor is used as a voltage controlled selector so that we can configure the connectivity of each RRAM cell. The major benefit of 1T1R is to avoid the sneak-path in the direct-connected structure. In Figure 2.14(b), we use one signal to control the transistors connecting to the same WL. It is clear that we only need to apply voltage on the target WL_i , BL_j , and only put $V_{G,i}$ to high, only the target cell can be configured. The metric of this structure can be summarized as follows:

- Low requirement on the non-linearity of RRAM device because voltage is only applied on the target cell.
- The controller design is much more easier than the direct-connected structure.

However, the 1T1R structure still has a disadvantage that the crossbar array area depends on the transistor, which is not scalable compared to RRAM device. Therefore, it is not a good option when the size of RRAM-crossbar is large.

2.2.2.3 One-Selector-One-RRAM (1S1R)

Here we proposed sneak-path-free digital RRAM-crossbar is shown in Figure 2.14(c). In this architecture, every drift-RRAM is connected in serial with a diffusive-RRAM as selector. Compared to the previous two implementations, the one-selector-one-RRAM (1S1R) can remain good scalability and eliminate sneak-path without additional controllers. Compared to [48], the R_{on} of selector can only affect little to the operation

Figure 2.14: Comparison of RRAM-crossbar implementation among (a) direct connection with sneak path (b) 1T1R with extra control signals (c) proposed sneak-path-free 1S1R without extra control signals; (d) comparison of analog and digital interfaces

on digital RRAM. Since $R_{off} \gg R_{HRS}$ as shown in Figure 2.10, the static resistance of 1S1R is almost equal to R_{off} . In addition, the majority of applied voltage is on the selector device when it is OFF. R_{on} Since each sneak-path will connect to at least three 1S1Rs, the operating voltage V only needs to satisfy the following equation to eliminate the sneak-path:

$$3V_{th,on} > V > V_{th,on} \quad (2.11)$$

We will further discuss about the simulation results regarding the sneak-path-free effect in Section 6.

2.3 RRAM Based Oscillator

Oscillatory behavior in RRAM device has been observed in [49, 50], and it is performed by the forming process of a drift-type RRAM. The recent diffusive-type RRAM [35, 36] shows the feasibility as oscillators. The RRAM based oscillator can be built as shown in Figure 2.15(a). In this RRAM based oscillator, a physical resistor R_s is in series with

Figure 2.15: (a) RRAM based oscillator schematic (b) Simulation result of oscillator (c) Discrete Fourier transform (DFT) analysis of oscillator

the RRAM device. The output voltage will depend on the proportion of R_s and RRAM resistance R_M . With the charge/discharge of capacitor C_M , the output voltage V_{out} will be approaching $\frac{V_{DD}R_M}{R_M+R_s}$. To make the oscillation work, we have the following requirements for the RRAM device in the structure:

- At a relatively high voltage V_h , the resistance of the RRAM R_M decreases significantly to R_l .
- If we drop the voltage to a low value V_l , R_M will increase to R_h .

Therefore, we can build up the oscillation if all the parameters in Figure 2.15(a) are well-fitted, following

$$\frac{R_l}{R_l + R_s} < V_l < V_h < \frac{R_h}{R_h + R_s}, \quad (2.12)$$

where R_l and R_h are the low resistance and high resistance of the RRAM device. As a result, R_M and V_{out} cannot be in a static state with the condition stated in Equation

(2.12). When the RRAM turns from R_l to R_h or vice versa, a negative resistance effect is performed. Moreover, the oscillatory frequency is determined by the value of R_s . Here we build the RRAM model for oscillator according to [51], and we can observe the oscillation result in Figure 2.15(b). Figure 2.15(c) shows the Discrete Fourier transform (DFT) analysis of Figure 2.15(b). Figure 2.15(d) shows the oscillatory frequencies v.s. different R_s values. With higher R_s , there is less current passing through the oscillator such that the charge/discharge will be slower, resulting in lower oscillatory frequency.

Chapter 3

Machine Learning Algorithms

3.1 Support Vector Machine Based Machine Learning

Support vector machine (SVM) is a supervised learning model with associated learning algorithms that analyzes data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

In summary, SVM is a binary classifier that seeks the maximum-margin separating hyper-plane from training samples. The type of SVM may be different depending on the property of data. For data can be linearly separated, a linear SVM as shown in Figure 3.1(a) can be applied. Otherwise, if the data cannot be linearly separated, a non-linear SVM in Figure 3.1(b) is required.

As SVM only provides binary classification, it is impossible to classify multiple labels with one SVM. Therefore, multiple SVMs are needed for complex applications such as fingerprint recognition as shown in Figure 3.2. The test image needs to go through several classifiers before we can obtain the classification result. It is clear that every SVM needs to provide the correct classification in order to obtain the final label. Therefore, the precision of each SVM should be high.

Figure 3.1: Demonstration of (a) linear SVM; (b) non-linear SVM

Figure 3.2: Fingerprint recognition by multiple SVMs

3.2 Single Layer Feed-Forward Neural Network Based Machine Learning

3.2.1 Single Layer Feed-Forward Network

The current data analytics is mainly based on machine learning algorithm to build a model to correlate input data with targeted output. Features extraction are also performed to extract the key information for data analytics in Neural Network. Neural network is the common model to build [52], and usually has two computational phases: training and testing. In the training phase, the weight coefficients of the neural network model are determined by minimizing the error between the trial and the targeted using the training input data. In the testing phase, the neural network with determined coefficients is utilized for the classification of the new testing data.

However, the input data may be in high dimension with redundant information. To

Figure 3.3: (a) Single-layer neural network (b) General flow of neural network training and testing

facilitate the training, feature extraction is usually needed performed to represent the characteristic data with redundancy or dimension reduction.

To speed-up the training process, we tackle this challenge from two perspectives. Firstly, we propose a general incremental machine learning architecture with minimal tuning of parameters. The proposed single-layer feed-forward network (SLFN) is shown in Figure 3.3, which is mainly based on incremental least-squares solution. Secondly, we analyze the key complexity of each learning step and propose a hardware friendly algorithm to explore the parallelism with minimized hardware operational complexity.

3.2.1.1 Feature Extraction

In general, the feature of original data $\mathbf{X}$ can be extracted by projection,

$$\mathbf{X}' = \mathbf{R} \cdot \mathbf{X} \quad (3.1)$$

where $\mathbf{X}'$ is the extracted feature. The projection matrix $\mathbf{R}$ can be found with the use of principal/singular components, random embedding or convolution [53]. Matrix $\mathbf{R}$ is computed off-line and used for dimension reductions. We can treat $\mathbf{R}$ as a new basis to represent columns of $\mathbf{X}$; and remove those small values to minimize the total squared reconstruction error by

$$\|\mathbf{X}' - \mathbf{R} \cdot \mathbf{X}\|_2 \quad (3.2)$$

One can observe significant matrix-vector multiplications during the feature extraction as shown in (3.1).

We also introduce a different feature extraction approach here. The first step is to compress raw data with reduced complexity. Image patches represent small groups of nearby pixels, which are commonly used for image feature extraction. For example, a 6×6 square pixel patch is moving across the whole input image with a step of 1 pixel and further is encoded by unsupervised learned dictionary $\mathbf{D}$ to form patches based image representation. Pooling is performed by summing up the quarter of encoded patches to form image features for classifier. Here, our analysis is mainly based on single layer feed forward network (SLFN) but it can be also extended to linear classifiers such as least square support vector machine (LS-SVM).

The learning dictionary $\mathbf{D}$ is determined from sparse coding based on orthogonal matching pursuit (OMP) method, where the key complexity comes from incremental least-squares problem. After building the dictionary $\mathbf{D}$, encoding for image patch x is performed based on soft threshold α .

$$f_j = \{\max(0, \mathbf{D}^T x - \alpha), \max(0, -\mathbf{D}^T x - \alpha)\} \quad (3.3)$$

where $j \in [0, k]$ and n is the number of basis of learning dictionary $\mathbf{D}$. α is a fixed threshold and $\{f_0, f_1 \dots f_k\}$ forms a patch in image representation. Therefore, in the feature learning, one can observe that matrix vector multiplication and sorting are the major computations.

3.2.1.2 Neural Network Based Learning

After feature extraction, one can perform various machine learning algorithms [54, 55, 14] for data analytics. As shown in Figure 3.3 for a typical neural network model, one needs to determine the network weights from training and then practice testing. We use n to represent the number of features with training input $X_f \in R^{N \times n}$. n is the training data size. The extracted feature will be input to the neural network with following relationship for the first layer output $\mathbf{preH}$:

$$\mathbf{preH} = \mathbf{X}_f \mathbf{A} + \mathbf{B}, \mathbf{H} = g(\mathbf{preH}) = \frac{1}{1 + e^{-\mathbf{preH}}} \quad (3.4)$$

where $\mathbf{A} \in \mathbb{R}^{n \times L}$ and $\mathbf{B} \in \mathbb{R}^{N \times L}$ are randomly generated input weight and bias formed by a_{ij} and b_{ij} between $[-1, 1]$; $\mathbf{H}$ is the hidden-layer output matrix generated from the Sigmoid function $g(\cdot)$ for activation.

The training of neural network is to minimize error with an objective function below

$$\min_{\mathbf{\Gamma}} \|\mathbf{H}\mathbf{\Gamma} - \mathbf{T}\|_2^2 + \eta \|\mathbf{\Gamma}\|_2^2 \quad (3.5)$$

where η is the regularized parameter and $\mathbf{T}$ is the label of training data.

One can solve Equation (3.5) either by iterative backward propagation method [56] or direct L2-norm solver method for least-squares problem [14]. The output weight can be obtained as $\|\tilde{\mathbf{H}}\mathbf{\Gamma} - \tilde{\mathbf{T}}\|$, and can be solved as

$$\mathbf{\Gamma} = (\tilde{\mathbf{H}}^T \tilde{\mathbf{H}})^{-1} \tilde{\mathbf{H}}^T \tilde{\mathbf{T}} = (\mathbf{H}^T \mathbf{H} + \eta \mathbf{I})^{-1} \mathbf{H}^T \mathbf{T}$$

$$\text{where } \tilde{\mathbf{H}} = \begin{pmatrix} \mathbf{H} \\ \sqrt{\eta} \cdot \mathbf{I} \end{pmatrix}, \quad \tilde{\mathbf{T}} = \begin{pmatrix} \mathbf{T} \\ \mathbf{0} \end{pmatrix} \quad (3.6)$$

Here $\tilde{\mathbf{H}} \in \mathbb{R}^{(N+L) \times L}$ is formed based on $\mathbf{H}$ and $\mathbf{I}$. For matrix $\mathbf{\Gamma}$, it is the solution of a least-square problem, where we adopt Cholesky decomposition to solve it [57]. We have also analyzed the major computations of Cholesky decomposition for least-square problem, which will be discussed in Section 6.

As a result, in the testing phase, output node $\mathbf{Y}$ is calculated by already determined hidden node value and output weight value as

$$\mathbf{Y} = \mathbf{H} \cdot \mathbf{\Gamma} \quad (3.7)$$

The index of the maximum value in $\mathbf{Y}$ represents the class that the test data belongs to.

Based on the computation analysis on feature extraction and neural network, we can observe that matrix-vector multiplication is the dominant operation as shown in Equation (3.1, 3.6, 3.7). As such, a hardware accelerator to facilitate the matrix-vector multiplication is indeed the critical requirement for the efficient machine-learning based data analytics.

3.2.1.3 Incremental Least-Square Solver Based Learning

The objective function in Equation (3.5) is a least-squares problem and can be solved using backwards propagations (BP) or direct solution based on matrix operations. Since our target is to have incremental learning with latest training samples, iterative gradient based backwards propagation is slow comparing to pseudo-inverse solutions [14], therefore, BP will not be elaborated in details. In fact, as discussed in the next sections, our proposed 3D multi-layer CMOS-RRAM architecture can accelerate the matrix-vector multiplications, which will also benefit BP based neural network training method.

Equation (3.6) shows how to obtain the output weight $\mathbf{\Gamma}$. The symmetric positive definite matrix $\tilde{\mathbf{H}}^T \tilde{\mathbf{H}}$ is decomposed into $\mathbf{Q}\mathbf{P}\mathbf{Q}^T$. $\mathbf{Q}$ is a lower triangular matrix with diagonal elements $q_{ii} = 1$ and $\mathbf{P}$ is a positive diagonal matrix. Such method can maintain

the same memory space as Cholesky factorization but need not perform square root extraction, as the square root of $\mathbf{Q}$ is resolved by diagonal matrix $\mathbf{P}$ [58]. Here, we use $\tilde{\mathbf{H}}_l$ to represent the matrix decomposition at l iteration where $l \leq L$ as below

$$\begin{aligned}\tilde{\mathbf{H}}_l^T \tilde{\mathbf{H}}_l &= [\tilde{\mathbf{H}}_{l-1} \ h_l]^T [\tilde{\mathbf{H}}_{l-1} \ h_l] \\ &= \begin{pmatrix} \tilde{\mathbf{H}}_{l-1}^T \tilde{\mathbf{H}}_{l-1} & \mathbf{v}_l \\ \mathbf{v}_l^T & g \end{pmatrix}\end{aligned}\quad (3.8)$$

where $(\mathbf{v}_l, g)$ is a new column generated from new hidden node output $h_l^T h_l$, compared to $\tilde{\mathbf{H}}_{l-1}^T \tilde{\mathbf{H}}_{l-1}$. Therefore, we can find

$$\begin{aligned}\mathbf{Q}_l \mathbf{P}_l \mathbf{Q}_l^T &= \begin{pmatrix} \mathbf{Q}_{l-1} & 0 \\ \mathbf{z}_l^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_{l-1} & 0 \\ 0 & p \end{pmatrix} \begin{pmatrix} \mathbf{Q}_{l-1}^T & \mathbf{z}_l \\ 0 & 1 \end{pmatrix}\end{aligned}\quad (3.9)$$

As a result, we can easily calculate the $\mathbf{z}_l$ and scalar p for Cholesky factorization as

$$\mathbf{Q}_{l-1} \mathbf{P}_{l-1} \mathbf{z}_l = \mathbf{v}_l, \ p = g - \mathbf{z}_l^T \mathbf{P}_{l-1} \mathbf{z}_l \quad (3.10)$$

where $\mathbf{Q}_l$ and $\mathbf{v}_l$ is known from Equation (3.8), which means we can continue use previous factorization result and update only according part. Please note that $\mathbf{Q}_1$ is 1 and $\mathbf{P}_1$ is $\tilde{\mathbf{H}}_1^T \tilde{\mathbf{H}}_1$.

As a conclusion, we have elaborated the basic learning on neural network and optimize Cholesky decomposition to solve the incremental least-squares problem. We have found the major computations are matrix-vector multiplications such as layer output in Equation (3.4, 3.7) and also Cholesky decomposition in Equation (3.8, 3.9, 3.10). Therefore, the proposed 3D multi-layer CMOS-RRAM architecture is designed to accelerate matrix-vector multiplication, which can be also extended to BP based training method, where matrix-vector operation is the major computation [59].

3.2.2 L2-Norm-Gradient Based Learning

3.2.2.1 Multi-Layer Neuron Network

A multi-layer neuron network based machine learning is known for many data analytic applications [60]. It typically has two computational stages: training stage and testing stage. During the training stage, features are extracted and a neuron network is built to map features to its class with trained input to determine the network coefficients. During the testing stage, based on the trained network, the new input data is classified

by choosing the closest label value of the class. The main computational bottleneck is actually from the training stage.

Given a neuron network with D inputs and M outputs shown in Figure 3.4, a training set $(x_1, t_1), (x_2, t_2), \dots, (x_N, t_N)$ is given with N number of training samples, D dimensional input features and M classes. There can be a number of hidden layers of neurons with a target error function in form of L2-norm. During the training stage, one needs to minimize the L2-norm error function with determined weights of A and W : the input weights and output weights respectively.

Usually, the hidden layer can be described with a nonlinear sigmoid activation function which maps all the values in the range $[-1, 1]$ calculated as:

$$\text{preH} = \mathbf{XA} + \mathbf{B}, \mathbf{H} = \frac{1}{1 + e^{-\text{preH}}} \quad (3.11)$$

where $\mathbf{A} \in \mathbb{R}^{D \times L}$ and $\mathbf{B} \in \mathbb{R}^{N \times L}$ are trained input weights and bias; $\mathbf{H} \in \mathbb{R}^{N \times L}$ is the sigmoid function for activation. The output of the neuron network $\mathbf{Y}$ is

$$\mathbf{Y} = \mathbf{HW}, \quad (3.12)$$

where $\mathbf{W} \in \mathbb{R}^{L \times N}$ is the trained output weight. The index with the maximum value will become the class during the testing stage.

The training objective function can be generally formulated a L2-norm minimization problem:

$$E = \|\mathbf{T} - \mathbf{Y}\|_2^2 \quad (3.13)$$

Therefore, the error objective E can be minimized using an iterative gradient-descent method [55] with the gradient of L2-norm respecting to each weight calculated as

$$\nabla E = \left(\frac{\partial E}{\partial a_{11}}, \frac{\partial E}{\partial a_{12}}, \dots, \frac{\partial E}{\partial a_{DL}}, \frac{\partial E}{\partial w_{11}}, \frac{\partial E}{\partial w_{12}}, \dots, \frac{\partial E}{\partial w_{LM}} \right) \quad (3.14)$$

Each weight can be updated as

$$a_{dl} = a_{dl} - \beta * \frac{\partial E}{\partial a_{dl}}, \quad d = 1, 2, \dots, D, \quad l = 1, 2, \dots, L \quad (3.15)$$

where β is the learning constant that defines the step length of each iteration in the negative gradient direction. As a summary, the L2-norm-gradient based machine learning training/testing algorithm can be observed in Algorithm 1.

Figure 3.4: L2-norm-gradient based multi-layer neural network for machine learning

Algorithm 1 Backpropagation training for neural network

Input: Training matrix $X \in \mathbb{R}^{N \times D}$, target matrix $T \in \mathbb{R}^{N \times M}$, number of hidden nodes L and learning rate β

Output: Neuron Network input weight A and output weight W

- 1: Initial guess: A^0 and W^0 ; Initial set $k = 1$, learning rate β
 - 2: Perform feed-forward operation, the input X is fed into the network as Eq. 3.11 and Eq. 3.12.
 - 3: Calculate training error on each neuron $E^{k-1} = ||\mathbf{T} - \mathbf{Y}^{k-1}||_2^2$
 - 4: Calculate gradient $\frac{\partial E^{k-1}}{\partial a_{ij}^{k-1}}$ and $\frac{\partial E^{k-1}}{\partial w_{ij}^{k-1}}$
 - 5: Update a_{ij}^{k-1} and w_{ij}^{k-1}

$$a_{dl}^k = a_{dl}^{k-1} - \beta * \frac{\partial E^{k-1}}{\partial a_{dl}^{k-1}},$$

$$w_{lm}^k = w_{lm}^{k-1} - \beta * \frac{\partial E^{k-1}}{\partial w_{lm}^{k-1}}$$
 - 6: Calculate the training error E^k with the updated weights
 - 7: If $E^k - E^{k-1}$ is larger than threshold, repeats from Step 3 and $k = k + 1$.
-

3.2.2.2 Direct-Gradient Based L2-Norm Optimization

As shown in Algorithm 1, it requires to compute the partial derivative of each weight using one forward pass through the network, following by one backward pass of the network [55]. Such a method is very efficient when the derivative of each weight is known. However, for a hardware based training, the derivative is usually unknown priori. In this algorithm, based on the L2-norm calculation, we show that a direct-gradient calculation can be realized on hardware as follows:

$$\frac{\partial E}{\partial a_{dl}} \approx \frac{E(a_{dl} + \Delta) - E(a_{dl})}{\Delta} \quad (3.16)$$

Figure 3.5: Diagram of simple deep learning with full-connect layers

where Δ is a small positive change comparing to a_{dl} . This idea is simple and accurate enough for the hardware evaluation of the L2-norm based error functions. In the following sections, we will show that by utilizing a coupled oscillator network and crossbar structures in RRAM devices, forward passing through the neuron network and also the evaluation of the L2-norm error function can be extremely fast with possibly done in one clock cycle.

3.3 Deep Convolutional Neural Network Based Machine Learning

3.3.1 Deep Learning for Multi-Layer Neural Network

Although the single-layer feed-forward network (SLFN) can achieve a desirable accuracy with very little time, it has not been proved to handle the complex benchmarks. For example, the recent research in [61] only shows the performance on UCI dataset [62], which is still far away from the real-life problems. The study in [63] shows that deep learning can be applied to train a neural network with multiple layers. A simple deep learning diagram is shown in Figure 3.5. In this diagram, there are $(n + 1)$ weight matrices connecting with adjacent nodes. When the output of neural network is obtained, we can update all the weights in the multi-layer neural network.

The architectures of deep learning are various including deep neural networks, deep convolutional neural networks, deep belief networks and recurrent neural networks. In the followings, we will detail the deep convolutional neural network with binary weights and neurons.

Figure 3.6: Example of a layer in image classification by convolutional neural network (CNN).

3.3.2 Convolutional Neural Network

Figure 3.5 illustrate a multi-layer neural network with all full-connect layers. However, usually the simple full-connect layers cannot extract enough information from input signals and hidden nodes. Therefore, the basic structure of convolutional neural network (CNN) [64] is applied in [65] and the deep convolutional neural network (DCNN) is formed. In this architecture, convolution layers are used to replace the full-connect layers in Figure 3.5. In each layer, there are several convolution cores to extract different information of one input pattern.

Figure 3.6 shows a layer of image classification by CNN with a 3×3 convolution core as an example. The dimension of input feature is 10×8 and all the pixels are real-valued. In this example, we need to perform the multiplication between two 9×1 vectors for 63 times. For the whole algorithm, there are many convolution cores and layers so that the computation complexity is huge. As such, the binary convolutional neural network (BCNN) is developed to lower the bit width in the operation.

3.3.3 Binary Convolutional Neural Network (BCNN)

As discussed in Section 3.3.2, the traditional direct mapping of CNN to hardware would require large and unnecessary hardware resource with poor parallelism. The recent work in [66] suggests a CNN using binary constraints during training. In this section, we will discuss how to generate a CNN model with bitwise parallelism for convolution, batch

normalization, pooling function and activation function.

3.3.3.1 Bitwise Convolution

The convolution is the most time-consuming and computation-intensive operation in CNN. The binary convolution in [66] uses $\{-1, +1\}$ for both input features and weights, so that the floating-point matrix-vector-multiplication operation is not required. However, the negative binary weights cannot be directly realized on hardware. To avoid the negative weights, recent work [67] uses bitwise XNOR and bit-count operation in $\{0, 1\}$ for hardware implementation. The bitwise CNN (BCNN) can be represented as follows:

$$s_k(x, y, z) = \sum_{i=1}^{w_k} \sum_{j=1}^{h_k} \sum_{l=1}^{D_k} w_k^b(i, j, l, z) \otimes a_{k-1}^b(i+x-1, j+y-1, l), \quad (3.17)$$

where $w_k^b \in \{0, 1\}^{w_k \times h_k \times D_{k-1} \times D_k}$ and $a_{k-1}^b \in \{0, 1\}^{W_{k-1} \times H_{k-1} \times D_{k-1}}$ are the binary weight in k -th block and the binary input feature map, respectively. s_k is the output of the binary convolution, and $\otimes$ is defined as bitwise XNOR operation. Comparing to a real-valued CNN in the single-precision data format, since the elements of weights and feature maps can be stored in 1-bit, both of the logic and memory resources required for binary convolutional (BinConv) layer can be greatly reduced. Meanwhile, it can lead to a higher parallelism as well as greater energy-efficiency improvement.

3.3.3.2 Bitwise Batch Normalization

Next, batch normalization is required to stabilize and accelerate the training process. In the inference stage, the normalization is retained to match training process. The output of the normalization can be represented by

$$a_k(x, y, z) = \frac{s_k(x, y, z) - \mu(x, y, z)}{\sqrt{\sigma^2(x, y, z)}} \gamma(x, y, z) + \beta(x, y, z), \quad (3.18)$$

where $\mu \in \mathbb{R}^{W_k \times H_k \times D_k}$ and $\sigma^2 \in \mathbb{R}^{W_k \times H_k \times D_k}$ are the expectation and variance over the mini-batch, while $\gamma \in \mathbb{R}$ and $\beta \in \mathbb{R}^{W_k \times H_k \times D_k}$ are learnable parameters [68] that scale and shift the normalized value. In the inference stage, μ , σ^2 , γ and β are all fixed to normalize the convolution output.

3.3.3.3 Bitwise Pooling and Activation Functions

The pooling layer performs a down-sampling across a $M \times M$ contiguous region on the feature map output by normalization layer. Pooling is used for selecting the most significant information from the features. It also provides translation invariance and reduces the computation intensity. Two kinds of pooling schemes are commonly used in CNN. One is the max-pooling, which takes the maximum value of the pooling region. The other is average-pooling, which takes the mean value of the pooling region. In this work, the binary max-pooling is applied with the following equation:

$$a'_k(i, j) = \text{Max}\{a_k(Mi + k, Mj + l)\}, (k, l \in (0, M)), \quad (3.19)$$

where a_k and a'_k are the features before and after pooling, respectively. The activation function to process the output of max-pooling is called binarization (Binrz), which can be represented as

$$a_k^{b'}(i, j) = \begin{cases} 0, & a'_k(i, j) \leq 0 \\ 1, & \text{otherwise} \end{cases} \quad (3.20)$$

To summarize pooling and binarization, we can observe that these two steps are to find out the sign of the maximum number in the pooling region. As such, we can do the binarization for all the numbers in the pooling region first, and then find the maximum among them. Since the results from binarization become only 0 or 1, the pooling process only needs to detect if there is any 1 in the region. An example with the exchange of pooling and binarization for a 2×2 region is shown below:

$$\begin{bmatrix} +0.70 & -0.42 \\ -0.21 & +0.35 \end{bmatrix} \Rightarrow \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow 1 \\ 0.70 \Rightarrow 1 \end{cases} \quad (3.21)$$

Here, the 2×2 real-value matrix denotes the output of batch normalization. The first path in Equation (3.21) is doing binarization first, then max-pooling, while the second path is doing it conversely. It is clear that exchanging the max-pooling and binarization will not affect the final output in the inference stage.

3.3.3.4 Bitwise CNN Model Overview

The overall working flow of a bitwise-parallelized CNN model is shown in Figure 3.7(b). Each BinConv layer takes the binary feature map generated from the previous layer as input and conducts a bitwise convolution between binary feature maps and binary

Figure 3.7: Computation flow comparison between (a) conventional CNN; and (b) bitwise-parallelized CNN.

filter weights. The convolution output is further processed by the normalization layer before down-sampling by a the max-pooling layer. The down-sampled feature maps are subsequently fed into Binrz layer that produces binary non-linear activations according to the input sign.

We summarize the training flow in Algorithm 2. In the training stage, all the weights are stored in real-valued format w_k and binarized to w_k^b during the forward pass (line-1 to -7). After the forward pass, the error signal C is generated by comparing the output of

Algorithm 2 Training flow of BNN. L is the number of layers, $\text{Binrz}(\cdot)$ defines the binarization, $\text{BinConv}(\cdot)$ defines the binary convolution, $\text{BatchNorm}(\cdot)$ defines the normalization and $\text{MaxPool}(\cdot)$ defines the 2×2 max-pooling.

```

0: {Forward pass:}
1: for  $k = 1$  to  $L$  do
2:    $w_k^b = \text{Binrz}(w_k)$ 
3:    $s_k = \text{BinConv}(a_{k-1}^b, W_k^b)$ 
4:    $a_k = \text{BatchNorm}(s_k)$ 
5:    $a_k = \text{MaxPool}(a_k)$ 
6:    $a_k^b = \text{Binrz}(a_k)$ 
7: end for
8:  $\hat{Y} = \text{softmax}(a_L)$ 
9:  $C = Y - \hat{Y}$ 
10:  $g_{a_L} = C$ 
11:  $g_{s_L} = \text{BackBatchNorm}(g_{a_L})$ 
12:  $g_{a_{L-1}^b} = g_{s_L} * w_L^b$ 
12: {Backward pass:}
13: for  $k = L - 1$  to  $1$  do
14:    $g_{a_k} = g_{a_k^b} \circ \mathbf{1}_{|a_k| \leq 1}$ 
15:    $g_{s_k} = \text{BackBatchNorm}(g_{a_k})$ 
16:    $g_{w_k^b} = g_{s_k} * a_{k-1}^b$ 
17:    $g_{a_{k-1}^b} = g_{s_k} * w_k^b$ 
18: end for
18: {Weights update:}
19: for  $k = 1$  to  $L$  do
20:    $w_k = w_k + \eta g_{w_k^b}$ 
21:    $w_k = \max(\min(w_k, 1), -1)$ 
22: end for

```

the network $\hat{Y}$ with the corresponding label Y (line-9). Then the error signal C is back-propagated to each previous layer to compute the weight's gradient $g_{w_k^b}$ and feature's gradient $g_{a_k^b}$ (line-13 to -18). Note that instead of using the real-valued weights w_k to compute feature's gradient of previous layer the binarized weight is applied for gradient computation, which is indicated at line-17. On the other hand, the value of w_k is constrained in the range of -1 to 1 at each iteration to ensure the convergence (line-21). It is different from the conventional update scheme that directly updates the weights without limiting their amplitudes.

The key difference between the developed bitwise CNN and the direct-truncated CNN using less precision bit [69] is illustrated as follows. The direct-truncated CNN is obtained by reducing the numerical precision in post-training phase while the bitwise CNN developed here is obtained by training with binary constraints [66]. As such,

the direct-truncated CNN that suffers from accuracy loss in general, but the bitwise CNN retains most of the accuracy with lowest precision. In the inference stage, only the binarized weights w_k^b will be retained for much smaller storage, faster inference, higher parallelism as well as higher energy-efficiency. The BNN algorithm is detailed in supplementary file.

3.4 Tensorized Neural Network Based Machine Learning

Previous neural network compression is simply performed by either precision-bit truncation or low-rank approximation [70, 71, 72, 73], which cannot maintain good balance between network compression and network accuracy. By representing dense data in high dimensional space with natural sparsity, tensorized data formatting can significantly compress the neural network complexity without much accuracy loss [74]. In this section, we discuss a tensor-train [75] formatted neural network during the training. The tensor-train based decomposition and compression will be first introduced. Then, a tensorized neural network (TNN) will be discussed based on the extension of general neural network. Finally, a layer-wise training of TNN using modified alternating least-squares method is further proposed. For simplicity, we illustrate the idea based on a single hidden layer neural network and later extend it to a multi-layer neural network. A tensor-train based neural network training method is also developed with the consideration of model compression using modified alternating least squares (MALS) method with complexity analysis provided.

3.4.1 Tensor-Train Decomposition and Compression

Tensors are natural multi-dimensional generation of matrices. Here, we refer one-dimensional data as *vectors*, denoted as $\mathbf{v}$. Two-dimensional arrays are *matrices*, denoted as $\mathbf{V}$ and higher dimensional arrays are *tensors* denoted as $\mathcal{V}$. To refer one specific element from a tensor, we use calligraphic upper letters $\mathcal{V}(\mathbf{i}) = \mathcal{V}(i_1, i_2, \dots, i_d)$, where d is the dimensionality of the tensor $\mathcal{V}$. We can effectively reshape a 2-dimensional matrix into a 4-dimensional tensor as shown in Figure 3.9.

A d -dimensional $n_1 \times n_2 \times \dots \times n_d$ tensor $\mathcal{V}$ is decomposed into the tensor-train data

format if tensor core $\mathbf{G}_k$ is defined as $r_{k-1} \times n_k \times r_k$ and each element is defined [75] as

$$\mathcal{V}(i_1, i_2, \dots, i_d) = \sum_{\alpha_0, \alpha_1, \dots, \alpha_d}^{r_0, r_1, \dots, r_d} \mathbf{G}_1(\alpha_0, i_1, \alpha_1) \mathbf{G}_2(\alpha_1, i_2, \alpha_2) \dots \mathbf{G}_d(\alpha_{d-1}, i_d, \alpha_d) \quad (3.22)$$

where α_k is the index of summation, which starts from 1 and stops at rank r_k . $r_0 = r_d = 1$ is for the boundary condition and $n_1, n_2, \dots, n_d$ are known as mode size. Here, r_k is the core rank and $\mathbf{G}$ is the core for this tensor decomposition. By using the notation of $\mathbf{G}_k(i_k) \in \mathbb{R}^{r_{k-1} \times r_k}$, we can rewrite the above equation in a more compact way:

$$\mathcal{V}(i_1, i_2, \dots, i_d) = \mathbf{G}_1(i_1) \mathbf{G}_2(i_2) \dots \mathbf{G}_d(i_d) \quad (3.23)$$

where $\mathbf{G}_k(i_k)$ is an $r_{k-1} \times r_k$ matrix, a slice from the 3-dimensional matrix $\mathbf{G}_k$.

Such a representation is memory-efficient to store high-dimensional data and hence with significant energy saving as well. For example, a d -dimensional tensor requires $N = n_1 \times n_2 \times \dots \times n_d = n^d$ number of parameters. However, if it is represented using the tensor-train format, it takes only $\sum_{k=1}^d n_k r_{k-1} r_k$ parameters. So if we manage to reduce the rank of each core, we can efficiently represent data with high compression rate and store them distributively. By permuting neural network weight into high dimensional data, we can compress the neural network. Here, we define a tensorized neural network (TNN) if the weight of the neural network can be represented in the tensor-train data format. For example, a two-dimensional weight $\mathbf{W} \in \mathbb{R}^{L_0 \times L_1}$ can be reshaped to a $k_1 + k_2$ dimensional tensor $\mathcal{W} \in \mathbb{R}^{l_{0,1} \times l_{0,2} \times \dots \times l_{0,k_1} \times l_{1,1} \times l_{1,2} \times \dots \times l_{1,k_2}}$ by factorizing $L_0 = \prod_{m=1}^{k_1} l_{0,m}$ and $L_1 = \prod_{m=1}^{k_2} l_{1,m}$ and such tensor can be decomposed into the tensor-train data format to save storages¹.

3.4.2 Tensor-Train Based Neural Network (TNN)

To make TNN clear, we first start with a general feed-forward neural network and then extend it to the tensor-train based neural network. We use a single hidden layer neural network as an example and the same principle can be applied to the multi-layer neural network [77, 78, 79]. Generally, we can train a neural network based on data features $\mathbf{X}$ and labels $\mathbf{T}$ with N_t number of training samples, N dimensional input features and M classes. During the training, one needs to minimize the error function with determined weights: $\mathbf{W}_1$ (at input layer) and $\mathbf{W}_2$ (at output layer) for a single hidden layer neural

¹Interested readers can also refer to [75, 76] for more details on the tensor-train data format.

Figure 3.8: Block matrices and their representation by a 4-dimensional tensor

network²:

$$E = ||\mathbf{T} - f(\mathbf{W}_1, \mathbf{W}_2, \mathbf{X})||_2 \quad (3.24)$$

where $f(\cdot)$ is the trained model to perform the predictions from input.

Here, we mainly discuss the inference (testing) process and leave the training process to the next section. The output of each layer is based on matrix multiplication and activation. For example, the first layer output $\mathbf{H}$ is

$$\mathbf{preH} = \mathbf{X}_t \mathbf{W}_1 + \mathbf{B}_1, \mathbf{H} = \frac{1}{1 + e^{-\mathbf{preH}}} \quad (3.25)$$

where $\mathbf{X}_t$ is the testing data. $\mathbf{W}_1 \in \mathbb{R}^{N \times L_1}$ and $\mathbf{B}_1 \in \mathbb{R}^{N_t \times L_1}$ are the input weights and bias respectively. Then the neural network output for a single hidden layer neural network is

$$\begin{aligned} \mathbf{Y} &= f(\mathbf{W}_1, \mathbf{W}_2, \mathbf{X}_t) \\ p(i/y_i) &\approx y_i, y_i \in \mathbf{Y} \end{aligned} \quad (3.26)$$

where i represents class index $i \in [1, M]$. We approximate the prediction probability for each class by the output of neural network.

For the tensor-train based neural network, Figure 3.9 shows the general idea. A two-dimensional weight is folded into a three-dimensional tensor and then decomposes into tensor cores $\mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_d$. These tensor cores are relative small matrices due to the small value of rank r leading to a high neural network compression rate. Then the whole neural network will be trained in the tensor-train data format.

The TNN inference is a directly application of the tensor-train-matrix-by-vector op-

²We ignore bias for a clear explanation.

Figure 3.9: Neural network weight tensorization and represented by tensor-train data format for parameter compression (from n^d to dnr^2)

erations [74, 75]. We will use $\mathbf{W} \in \mathbb{R}^{N \times L}$ to discuss the forward pass of neural network. Firstly, we rearrange $\mathbf{W}$ to a d -dimensional tensor $\mathcal{W}$ whose k_{th} dimension is a vector of length $n_k l_k$. Here, we define n_k and l_k as $N = \prod_{k=1}^d n_k$ and $L = \prod_{k=1}^d l_k$. Without consideration of the bias $\mathbf{B}$ and activation function, the neural network forward pass $\mathbf{H} = \mathbf{XW}$ in the tensor-train data format is

$$\mathcal{H}(\mathbf{i}) = \sum_{\mathbf{j}=[j_1, j_2, \dots, j_d]}^{l_1, l_2, l_3, \dots, l_d} \mathcal{X}(\mathbf{j}) \mathbf{G}_1[i_1, j_1] \mathbf{G}_2[i_2, j_2] \dots \mathbf{G}_d[i_d, j_d] \quad (3.27)$$

where $\mathbf{i} = i_1, i_2, \dots, i_d, i_k \in [1, n_k]$, $\mathbf{j} = j_1, j_2, \dots, j_d, j_k \in [1, l_k]$ and $\mathbf{G}_k[i_k, j_k] \in \mathbb{R}^{r_{k-1} \times r_k}$ is a slice of cores. We use a pair $[i_k, j_k]$ to refer an index of vector $[1 \ n_k l_k]$, where $\mathbf{G}_k \in \mathbb{R}^{r_{k-1} \times n_k l_k \times r_k}$. Since the fully-connected layer is a special case of convolutional layer with kernel size 1×1 , such tensorized weights can also be applied to other convolutional layers.

This tensor-train-matrix-by-vector multiplication complexity is $O(dr^2 n_m \max(N, L))$ [74], where r is the maximum rank of cores $\mathbf{G}_i$ and n_m is the maximum mode size $m_k n_k$ of tensor $\mathcal{W}$. This can be very efficient if the rank r is very small compared to general matrix-vector multiplication. It is also favorable for distributed computation on RRAM devices since each core is small and matrix multiplication is associative.

Chapter 4

In-Memory Machine Learning Architecture

4.1 RRAM Network Based Logic Operations

In this chapter, we will introduce two types of RRAM networks. RRAM-crossbar network can be used as matrix-vector multiplication accelerator and we will illustrate the detailed mapping. The coupled RRAM oscillator network can be applied for low-power and high-throughput L2-norm calculation.

4.1.1 RRAM-Crossbar Network

In this section, we implement matrix-vector multiplication on the binary RRAM-crossbar. It is one always-on operation in various data-analytic applications such as compressive sensing, machine learning. For example, the feature extraction can be achieved by multiplying Bernoulli matrix in [80].

Matrix multiplication can be denoted as $Y = \Phi X$, where $X \in \{0, 1\}^{N \times P}$ and $\Phi \in \{0, 1\}^{M \times N}$ are the multiplicand matrices, and $Y \in \mathbb{Z}^{M \times P}$ is the result matrix.

4.1.1.1 Mapping of RRAM-Crossbar for Matrix-Vector Multiplication

In this section, hardware mapping of matrix multiplication on the proposed architecture is introduced. The logic required is a matrix-vector multiplier by the RRAM-crossbar. Generally, we utilize current merge on bitlines to realize matrix-vector multiplication, than use an RRAM-based flash ADC for format converting. Here, a three-step RRAM-crossbar based binary matrix-vector multiplier is proposed, in which both the input and output of the RRAM-crossbar are binary data without the need of ADC. The three

RRAM-crossbar step: parallel digitizing, XOR and encoding are presented in details as follows. As the output of an RRAM-crossbar array can be connected to the input of another RRAM-crossbar array, we can use multiple RRAM arrays in the logic block for the mappings. Here we use symbol s to denote the result of binary matrix-vector multiplication. Therefore, s follows

$$0 \leq s \leq N, \quad (4.1)$$

where N is the maximum result. To illustrate the three-step procedure more clearly, we will use the following matrix-vector multiplication as an example:

$$[00101011] \times [10111110]^T = 3 \quad (4.2)$$

The output after the three-step procedure will be shown when $s = 3$ and $N = 8$.

Parallel Digitizing The first step is called parallel digitizing, which requires $N \times N$ RRAM-crossbars. The idea is to split the matrix-vector multiplication to multiple inner-product operations of two vectors. Each inner-product is produced by one RRAM-crossbar. For each crossbar, as shown in Figure 4.1, all columns are configured with same elements that correspond to one column in random Boolean matrix Φ , and the input voltages on word-lines (WLs) are determined by x . As $g_{on} \gg g_{off}$ (normally $\frac{g_{on}}{g_{off}} > 100$), current on RRAMs with high impedance are insignificant, so that the voltages on BLs

Figure 4.1: Parallel digitizing step of RRAM-crossbar in matrix multiplication

approximately equal to $kV_{r\text{gon}}R_s$ according to Equation (2.9) where k is the number of RRAM with in low-resistance state (g_{on}). Here we assume that the number of columns is less than 100 because the ON/OFF ratio is finite.

It is obvious that voltages on bit-lines (BLs) are all identical. Therefore, the key to obtain the inner-product is to set ladder-type sensing threshold voltages for each column,

$$V_{th,j} = \frac{(2j+1)V_{r\text{gon}}R_s}{2}, \quad (4.3)$$

where $V_{th,j}$ is the threshold voltage for the j_{th} column. The $O_{i,j}$ is used to denote the output of column j in RRAM-crossbar step i after sensing. Therefore, for the output we have

$$O_{1,j} = \begin{cases} 1, & j \leq s \\ 0, & j > s, \end{cases} \quad (4.4)$$

where s is the inner-product result. In other words, the first $(N-s)$ output bits are 0 and the rest s bits are 1 ($s \leq N$). In our example, $x_i = [00101011]$ and $\phi_i = [10111110]$, and the corresponding output $O_1 = [11100000]$.

Figure 4.2: XOR step of RRAM-crossbar in matrix multiplication

XOR The inner-product output of parallel digitizing step is determined by the position where $O_{1,j}$ changes from 0 to 1. The XOR takes the output of the first step, and performs XOR operation for every two adjacent bits in $O_{1,j}$, which gives the result index. For the same example of $s = 3$, we need to convert the first-step output $O_1 = [11100000]$ to $O_2 = [00100000]$. The XOR operation based on RRAM-crossbar is shown in Figure

4.2. According to parallel digitizing step, $O_{1,j}$ must be 1 if $O_{1,j+1}$ is 1. Therefore, XOR operation is equivalent to the AND operation $O_{1,j} \oplus O_{1,j+1} = O_{1,j} \overline{O_{1,j+1}}$, and therefore we have

$$\overline{O_{2,j}} = \begin{cases} \overline{O_{1,j}} + O_{1,j+1}, & j < N-1 \\ \overline{O_{1,j}}, & j = N-1. \end{cases} \quad (4.5)$$

In addition, the threshold voltages for the columns have to follow

$$V_{th,j} = \frac{V_{rgon} R_s}{2} \quad (4.6)$$

Equation (4.5, 4.6) show that only output of s_{th} column is 1 on the second step, where s is the inner product result. Each crossbar in XOR step has the size of $N \times (2N-1)$.

Encoding The third step takes the output of XOR step and produces s in binary format as an encoder. Therefore, O_3 should be in the binary format of s . In our example, $O_3 = [00000011]$ when $s = 3$. In the output of XOR step, as only one input will be 1 and others are 0, according binary information is stored in corresponding row, as shown in Figure 4.3. Encoding step needs $N \times n$ RRAMs, where $n = \lceil \log_2 N \rceil$ is the number of bits in order to represent N in binary format. The thresholds for the encoding step are set following Equation (4.6) as well.

Figure 4.3: Encoding step of RRAM-crossbar in matrix multiplication

Figure 4.4: RRAM-based inner-product operation

Adding and Shifting for Inner-product Result The output of encoding step is in binary format, but some processes are needed to obtain the final inner-product result. Adder and shifter are designed to complete this process as shown in Figure 4.4. We suppose the original data is 8-bit and data dimension is 512, the workload of adder is 512 without any acceleration. With three-step RRAM-crossbar accelerator as pre-processing, the workload of adder can be significantly reduced to 9 ($\log_2 512$). Detailed comparison results will be shown in Section 4.1.1.2.

4.1.1.2 Performance Evaluation

Experimental Application: Feature Extraction for Fingerprint Matching Feature extraction or sparse-representation is commonly applied such as in fingerprint image matching, which can be mapped with ($M \ll N$). This operation can minimize the volume of data to be stored in memory as well as reduce complexity in data analytics. In the following we show how to map this operation on the digitalized RRAM-crossbar. In such process, X is the original fingerprint image in high dimension with ($N \times P$) pixels, Φ a random Bernoulli matrix with ($M \times N$) for feature extract, and Y the features in low dimension with ($M \times P$) pixels. The according dimension reduction ratio γ is

$$\gamma = \frac{M}{N}. \quad (4.7)$$

In feature extraction of fingerprint image, the random Bernoulli matrix Φ is with fixed elements. Therefore, elements in Bernoulli matrix are stored in RRAMs of logic block, and original image as the input of logic accelerator.

Experimental Settings The hardware evaluation platform is implemented on a computer server with 4.0GHz core and 16.0GB memory. Feature extraction is implemented

Figure 4.5: Hardware Performance Scalability under Different Reduced Dimension for (a) area; (b) delay; (c) energy (d) EDP

by general processor, CMOS-based ASIC, non-distributed and distributed in-memory computing based on digitalized RRAM crossbar respectively. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as $1\text{k}\Omega$ and $1\text{M}\Omega$ as on-state and off-state resistance respectively according to [81]. The general processor implementation is based on Matlab simulation on computer server. A CMOS-based feature extraction design is implemented by Verilog in distributed architecture and synthesized with CMOS 65nm low power PDK. The working frequency of general processor implementation is 4.0GHz while the CMOS ASIC feature extraction design frequency is 1.0GHz. For in-memory computing based on the proposed RRAM crossbar, write voltage V_w is set as 0.8V and read voltage V_r is set as 0.1V as well as duration time of 5ns. In addition, the analog computation on RRAM-crossbar is performed for comparison based on design in [82].

Table 4.1: Performance comparison of matrix-vector multiplication under among software and hardware implementations.

Implementation		Frequency	Area	Time	Power	Energy
CPU (Matlab)		4.0GHz	$177mm^2$	$1.78ms$	84W	$0.1424J$
CMOS-ASIC		1.0GHz	$5mm^2$	$69.632\mu s$	34.938W	$2.4457mJ$
Digitized XIMA	RRAM	200MHz	$3.28mm^2$	$4.92\mu s$	4.096W	$20.15\mu J$
	CMOS		$128\mu m^2$	$1.311ms$	$100\mu W$	$0.131\mu J$
	Overall		$3.28mm^2$	$1.316ms$	4.096W	$20.16\mu J$
Distributed digitized XIMA	RRAM	200MHz	$0.05mm^2$	$4.92\mu s$	4.096W	$20.15\mu J$
	CMOS		$8192\mu m^2$	$20.48\mu s$	6.4mW	$0.131\mu J$
	Overall		$0.06mm^2$	$25.40\mu s$	4.102W	$20.28\mu J$
Distributed analogue XIMA	RRAM	200MHz	$0.05mm^2$	$1.64\mu s$	1.28W	$2.1\mu J$
	CMOS		$8.27mm^2$	$20.48\mu s$	6.4mW	$0.131\mu J$
	Overall		$8.32mm^2$	$22.12\mu s$	1.29W	$2.13\mu J$

General Performance Comparison In this section, 1,000 fingerprint images selected from [83] are processed as binarization images and stored in memory with 328×356 resolution. To agree with patch size, random Bernoulli $N \times M$ matrix is with fixed N and M of 356 and 64, respectively. The detailed comparison is shown in Table 4.1 with numerical results including energy consumption and delay obtained for one image on average of 1,000 images. Here the CMOS in XIMA refers to the memory controllers.

Among hardware implementations, in-memory computing based on the proposed XIMA achieves better energy-efficiency than CMOS-based ASIC. Non-distributed XIMA (only one data and logic block inside memory) needs fewer CMOS control bus but large data communication overhead on a single-layer crossbar compared to distributed RRAM crossbar. Although distributed analog RRAM crossbar can achieve the best in energy perspective but has larger area compared to the digitalized one. Shown in Table 4.1, RRAM crossbar in analog fashion only consumes $2.1\mu J$ for one vector multiplication while the proposed architecture requires $20.15\mu J$ because most of power consumption comes from RRAM in computing instead of ADCs. However, ADCs need more area so that RRAM crossbar with analog fashion is $8.32mm^2$ while the proposed one is only $0.05mm^2$ because of the high density of RRAM crossbar.

Calculation error of analog and digitalized RRAM crossbar are compared in Figure 4.6, where M and N are both set as 256. Here the calculation error is the result variation of two schemes: ADC with analog RRAM and sense amplifier with digital RRAM. The RRAM error rate refers to the variation of write voltage applied on the device. Calculation error is very low when RRAM error rate is smaller than 0.004 for both analog and digitalized fashion RRAM. However, when RRAM error rate reaches 0.01, calculation error rate of analog RRAM crossbar goes to 0.25, much higher than the

other one with only 0.07. As such, computational error can be reduced in the proposed architecture compared to analog fashion RRAM crossbar.

Scalability Study Hardware performance comparison among CMOS-based ASIC, non-distributed and distributed XIMA with varying M is shown in Figure 4.5. From area consumption perspective shown in Figure 4.5(a), distributed RRAM-crossbar is much better than the other implementations. With increasing M from 64 to 208, its total area is from $0.057mm^2$ to $0.185mm^2$, approximately 100x smaller than the other two approaches. Non-distributed RRAM crossbar becomes the worst one when $M > 96$. From delay perspective shown in Figure 4.5(b), non-distributed RRAM crossbar is the worst because it has only one control bus and takes too much time on preparing of computing. Delay of non-distributed RRAM crossbar grows rapidly while distributed RRAM crossbar and CMOS-based ASIC implementation maintains on approximately $21\mu s$ and $70\mu s$ respectively as the parallel design. For energy-efficiency side shown in Figure 4.5(c), both non-distributed and distributed RRAM crossbar do better as logic accelerator is off at most of time. The proposed architecture also performs the best in energy-delay product (EDP) shown in Figure 4.5(d). Distributed XIMA performs the best among all implementation under different specifications. The EDP is from $0.3 \times 10^{-9}s \cdot J$ to $2 \times 10^{-9}s \cdot J$, which is 60x better than non-distributed RRAM crossbar and 100x better than CMOS-based ASIC.

What is more, hardware performance comparison with varying N is shown in Figure 4.7. Area and energy consumption trend is similar to Figure 4.5. But for computational delay, the proposed architecture cannot maintain constantly as Figure 4.5(b) because it

Figure 4.6: Calculation error comparison between multi-leveled and binary RRAM

Figure 4.7: Hardware Performance Scalability under Different Original Dimension for (a) area; (b) delay; (c) energy (d) EDP

needs much time to configure the input, but still the best among the three. Distributed XIMA still achieves better performance than the other two.

4.1.2 Coupled RRAM Oscillator Network

4.1.2.1 Coupled-RRAM-Oscillator Network for L2-Norm Calculation

In a coupled oscillator network, we use the RRAM oscillator cell similar with Figure 2.15(a) (R_s replaced with a PMOS). By altering the gate voltage of the PMOS, we can control the oscillatory frequency as in Figure 2.15(d). Referring to previous works [84, 85], the proposed coupled-RRAM-oscillator network is shown in Figure 4.8(a). Here, oscillators are directly coupled with RRAMs. The oscillatory frequency of each oscillator is determined by multiple input voltages. As mentioned in Figure 2.15(d), the voltage-controlled PMOS can be viewed as a voltage-controlled resistor. Figure 4.8(b) and Figure 4.8(c) are the schematic of oscillators with one input and multiple inputs. Af-

ter directly coupling, we have an inverter and a feedback resistor with a diode-connected NMOS. Because of the rectifying property of diode connected NMOS, these elements guarantee that the current will only pass from the oscillator network to V_{out} . An RC circuit is design at V_{out} so that we can measure the DC voltage of V_{out} . The output voltage indicates the relative synchronization of the coupled oscillators.

Figure 4.8: (a) RRAM oscillator network schematic (b) Oscillator with single input (c) Oscillator with multiple input

Figure 4.9: Transient response of RRAM oscillator network output with (a) $C_{out} = 1pF$ (b) $C_{out} = 2pF$ (c) $C_{out} = 5pF$ (d) $C_{out} = 20pF$

Table 4.2: Detailed evaluation of RRAM based oscillator

Parameter	V_{DD}	I_{max}	I_{min}	Power	Frequency	Area
Value	1V	$9\mu A$	$3\mu A$	$5.87\mu W$	6MHz	$0.49\mu m^2$

4.1.2.2 Performance Evaluation

The detailed evaluation of RRAM based oscillator is shown in Table 4.2. Compared to the design in [84, 85], it has power consumption and can be controlled by multiple voltage inputs easily. As the current through RRAM based oscillator is low, the working frequency of the oscillator is lower than previous works.

As we discussed above, V_{out} is charged by the oscillator network. Therefore, the capacitance of C_{out} should be large enough to avoid the voltage drop at V_{out} . At the meantime, large capacitance leads to a large settling time. Figure 4.9 shows the transient response with $C_{out} = 1pF$, $2pF$, $5pF$ and $20pF$, respectively. Obviously $C_{out} = 5pF$ is the best configuration because we can capture a stable output at $7.5\mu s$. In this simulation, we set R_{out} as $10M\Omega$.

4.2 RRAM Network Based In-Memory Machine Learning Accelerator

In this chapter, we will introduce different computing accelerators based on RRAM devices. Firstly, the distributed RRAM-crossbar in-memory architecture (XIMA) will be illustrated in detail. The integration of RRAM-crossbar based storage and computing brings advantages on throughput. After that, two types of 3D CMOS-RRAM architectures will be introduced. The major merits of the 3D architecture are less area overhead as well as higher throughput by the 3D interconnection.

4.2.1 Distributed RRAM-Crossbar In-Memory Architecture (XIMA)

Conventionally, processor and memory are separate components that are connected through I/Os. With limited width and considerable RC-delay, the I/Os are considered the bottleneck of system overall throughput. As memory is typically organized in H-tree structure, where all leaves of the tree are data arrays, it is promising to impose in-memory computation with parallelism at this level. In the followings, we will propose a distributed RRAM-crossbar in-memory architecture (XIMA). Such architecture will be used for several machine learning applications in Section 6.

4.2.1.1 Memory-Computing Integration

Because both data and logic units have uniform structure when implemented on RRAM-crossbar, half of the leaves are exploited as logic elements and are paired with data arrays. The proposed architecture is illustrated in Figure 4.10. The distributed local data-logic pairs can form one local data path such that the data can be processed locally in parallel, without the need of being readout to the external processor.

Coordinated by the additional controlling unit called *in-pair control bus* the in-memory computing is performed in following steps. (1) logic configuration: processor issues the command to configure logic by programming logic RRAM-crossbar into specific pattern according to the functionality required; (2) load operand: processor sends the data address and corresponding address of logic accelerator input; (3) execution: logic accelerator can perform computation based on the configured logic and obtain results after several cycles; (4) write-back: computed results are written back to data array directly but not to the external processor.

With emphasis on different functionality, the RRAM-crossbars for data storage and logic unit have distinctive interfaces. The data RRAM-crossbar will have only one row activated at one time during read and write operations, and logic RRAM-crossbar;

Figure 4.10: Overview of distributed in-memory computing architecture on RRAM-crossbar

however, we can have all rows activated spontaneously as rows are used to take inputs. As such, the input and output interface of logic crossbar requires AD/DA conversions, which could outweigh the benefits gained. Therefore, in this work, we propose a conversion-free digital-interfaced logic RRAM-crossbar design, which uses three layers of RRAM crossbars to decompose a complex function into several simple operations that digital crossbar can tackle.

4.2.1.2 Communication Protocol and Control Bus

The conventional communication protocol between external processor and memory is composed of *store* and *load* action identifier, address that routes to different locations of data arrays, and data to be operated. With additional in-memory computation capacity, the proposed distributed in-memory computing architecture requires modifications on the current communication protocol. The new communication instructions are proposed in Table 4.3, which is called in-pair control.

In-pair control bus needs to execute instructions in Table 4.3. SW (store word) instruction is to write data into RRAMs in data array or in-memory logic. If target address is in data array, it will be a conventional write or result write-back; otherwise it will be logic configuration. LW (load word) instruction performs as conventional read operation. ST (start) instruction means to switch on the logic block for computing after the computation setup has been done. WT (wait) operation is to stop reading from instruction queue during computing.

Table 4.3: Protocols between external processor and control bus.

Instruction	Operation 1	Operation 2	Action	Function
SW	Address 1	Address 2	Address 1 data to Address 2	Store data, configure logic, in-memory results write-back
	Data	Address	Store Data to Address	
LW	Address	-	Read Data from Address	Standard read
ST	Block Index	-	Switch logic block on	Start in-memory computing
WT	-	-	Wait for logic block response	Halt while performing in-memory computing

Besides communication instructions, memory address format is also different from that in the conventional architecture. To specify a byte in the proposed architecture, address includes the following identifier segments. Firstly, the data-logic pair index segment is required, which is taken by block decoders to locate the target data-logic pair. Secondly, one-bit flag is needed to clarify that whether the target address is in data array or in-memory logic crossbar. Thirdly, if logic accelerator is the target, additional segment has to specify the layer index. Lastly, rest of address segment are row and column indexes in each RRAM-crossbar. An address example for data array and in-memory logic is shown in Figure 4.11.

Figure 4.11: Detailed structure of control bus and communication protocol.

Figure 4.12: (a) 3D CMOS-RRAM accelerator architecture; (b) RRAM based matrix-vector multiplication engine.

To perform logic operation, the following instructions are required to be performed. Firstly, we store the required input data and RRAM values with SW operation. Secondly, an ST instruction will be issued to enable all the columns and rows to perform the logic computing. The WT instruction is also performed to wait for the completion of logic computing. At last, LW instruction is performed to load the data from the output of RRAM-crossbar.

Given the new communication protocol between general processor and memory is introduced, one can design the according control bus as shown in Figure 4.11. The control bus is composed of an instruction queue, an instruction decoder, an address decoder and a SRAM array. As the operation frequency of RRAM-crossbar is slower than that of external processor, instructions issued by the external processor will be stored in the instruction queue first. They are then analyzed by instruction decoder on a first-come-first-serve (FCFS) basis. The address decoder obtains the row and column index from the instruction; and SRAM array is used to store temporary data such as computation results, which are later written back to data array.

4.2.2 3D XIMA

4.2.2.1 3D Single-Layer CMOS-RRAM Architecture

Recent work [86] has shown that the 3D integration supports heterogeneous stacking because different types of components can be fabricated separately, and layers can be stacked and implemented with different technologies. Therefore, stacking non-volatile memories on top of microprocessors enables cost-effective heterogeneous integration. Furthermore, works in [87, 88, 89] have also shown the feasibility to stack RRAM on

4.2.2.2 3D Multi-Layer CMOS-RRAM Architecture

Besides the 3D single-layer CMOS-RRAM architecture, we also discuss the 3D multi-layer CMOS-RRAM architecture for on-line machine learning applications as well. This 3D multi-layer CMOS-RRAM accelerator with three layers is shown in Figure 4.13(a). This accelerator is composed of a two-layer RRAM-crossbar and a one-layer CMOS circuit. As Figure 4.13(a) shows, layer 1 of RRAM-crossbar is implemented as a buffer to temporarily store input data to be processed. Layer 2 of RRAM-crossbar performs logic operations such as matrix-vector multiplication and also vector addition. The details of implementation will be introduced in Section 4. Note that buffers are designed to separate resistive networks between layer 1 and layer 2. The last layer of CMOS contains read-out circuits for RRAM-crossbar and performs as logic accelerators designed for other operations besides matrix-vector multiplication, including pipelined divider, look-up table (LUT) designed for division operation and activation function in machine learning.

Moreover, Figure 4.13(b) shows the work flow for incremental machine learning based on the proposed architecture. Firstly, detailed architecture of machine learning (ML) (e.g. number of layers and activation function) is determined based on the accuracy requirements and data characteristics. Secondly, operations of this machine learning algorithm are analyzed and reformulated so that all the operations can be accelerated in 3D multi-layer CMOS-RRAM architecture as illustrated in Figure 4.13(a). Furthermore, the bit-width operating on RRAM-crossbar is also determined by balancing the accuracy loss and energy saving. Finally, logic operations on RRAM-crossbar and CMOS are configured based on the reformulated operations, energy saving and speed-up.

Such a 3D multi-layer CMOS-RRAM architecture has advantages in three manifold. Firstly, by utilizing RRAM-crossbar for input data storage, leakage power of memory is largely removed. In a 3D architecture with TSV interconnection, the bandwidth from this layer to next layer is sufficiently large to perform parallel computation. Secondly, RRAM-crossbar can be configured as computational units for the matrix-vector multiplication with high parallelism and low power. Lastly, with an additional layer of CMOS-ASIC, more complicated tasks such as division and non-linear mapping can be performed. As a result, the whole training process of machine learning can be fully mapped to the proposed 3D multi-layer CMOS-RRAM accelerator architecture towards real-time training and testing.

The 3D multi-layer CMOS-RRAM architecture will be used for on-line machine learning (both learning and inference phases) in the followings. We will further introduce the detailed mapping and results in Section 5.2 and Section 6.2.

Chapter 5

Mapping Machine Learning on RRAM

In this chapter, we will show the mapping of several machine learning applications on different architectures discussed in Section 4. For each application, we will introduce the detailed mapping strategy, experimental settings and evaluation results.

5.1 Machine Learning Algorithms on XIMA

We will show the detailed mapping of several machine learning applications on distributed in-memory computing architecture (XIMA), including single-layer feed-forward neural network (SLFN) based learning, binary convolutional neural network (BCNN) based inferences on passive array as well as 1S1R array.

5.1.1 SLFN Based Learning and Inference Acceleration

The details of this work has been presented in [46]. In this work, we will map the SLFN based learning and inference process according to the algorithm and network in Section 3.2. In this network, matrix-vector multiplication, the major operation in machine learning on RRAM-crossbar is implemented by three steps with detailed mapping strategy for each step. Here, we will show the mapping of $\mathbf{X}'\mathbf{A}$ in Figure 3.3 as example.

In matrix-vector multiplication, we assume the output matrix is $\mathbf{Y}$. For better understanding, we replace $\mathbf{X}'$ with $\mathbf{X}$ and $\mathbf{A}$ with Φ , the equation becomes $\mathbf{Y} = \mathbf{X}\Phi$. For every element in $\mathbf{Y}$, it follows

$$y_{ij} = \sum_{k=1}^N x_{ik} \phi_{kj}, \quad (5.1)$$

where x and ϕ are the elements in $\mathbf{X}$ and Φ , respectively. We will show how to calculate y_{ij} by three-step RRAM-crossbar.

Figure 5.1: Detailed Mapping for digitalized matrix-vector multiplication (a) Parallel Digitizing (b) XOR 2 (c) Encoding (d) Legend of RRAM mapping

5.1.1.1 Step 1: Parallel Digitizing

The first step parallel digitizing requires an $N \times N$ RRAM-crossbar. The idea is to split the matrix-vector multiplication to multiple inner-product operations of two vectors. Each inner-product is produced RRAM-crossbar as shown in Figure 5.1(a). x is set as crossbar input and ϕ is written in RRAM cells. In the $N \times N$ RRAM-crossbar, resistance of RRAMs in each column are the same, but V_{th} among columns are different. We have

$$V_{th,j} = \frac{(2j+1)V_r R_{fixed}}{2R_{on}}, \quad (5.2)$$

which are ladder-like threshold voltages to identify the result. If the inner-product result is s , the output of columns will be

$$O_{1,j} = \begin{cases} 1, & j \leq s \\ 0, & j > s, \end{cases} \quad (5.3)$$

5.1.1.2 Step 2: XOR

The second step is XOR to identify the index of s where $O_{1,s} = 1$ and $O_{1,s+1} = 0$, so the operation is $O_{1,s} \oplus O_{1,s+1}$. According to (5.3), the case $O_{1,j} = 0$ and $O_{1,j+1} = 1$ is impossible, so the XOR operation can be converted to $O_{1,s} \overline{O_{1,s+1}}$ and $\overline{O_{1,s}} + O_{1,s+1}$. The mapping of RRAM-crossbar input and resistance in this step is shown in Figure 5.1(b), and threshold voltage configuration is

$$V_{th,j} = \frac{V_r R_{fixed}}{2R_{on}} \quad (5.4)$$

Figure 5.2: RRAM-based inner-product operation

Therefore, only the output of s th column is 1 and the others are 0.

5.1.1.3 Step 3: Encoding

The third step takes the output of XOR step and produces s in binary format as an encoder. The input of RRAM-crossbar is from the second step and only one input is 1, so that according binary format is stored in corresponding row, as shown in Figure 5.1(c). Encoding step needs an $N \times n$ RRAM-crossbar, where $n = \lceil \log_2 N \rceil$ is the number of bits in order to represent N in binary format. The thresholds for the encoding step are set following Equation (5.4) as well.

5.1.1.4 Step 4: Adding and Shifting for Inner-product Result

The output of encoding step is in binary format, but some processes are needed to obtain the final inner-product result. Adder and shifter are designed to complete this process as shown in Figure 5.1(d). We suppose the original data is 8-bit and data dimension is 512, the workload of adder is 512 without any acceleration. With three-step RRAM-crossbar accelerator as pre-processing, the workload of adder can be significantly reduced to 9 ($\log_2 512$).

5.1.2 BCNN Based Inference Acceleration on Passive Array

The details of this section has been presented in [90]. As indicated in previous works, RRAM-crossbar can be applied to perform logic operations, especially the dot-product based convolutions. Previous mapping of CNN are mainly based on the traditional analog RRAM-crossbar [43, 48]. The main limitation is there can exist the huge non-uniform analog resistance for undetermined states in the analog computation, which can result in convolution error.

In this work, we will map the bitwise convolutional neural network (BCNN) based inference process according to the algorithm and network in Section 3.3.3. We will focus on the mapping of all the bitwise CNN operations on the passive array based XIMA such as convolution, batch normalization, pooling and activation functions.

5.1.2.1 Mapping Bitwise Convolution

According to (3.17), the bitwise convolution can be split into several XNOR and bit-count results of two vectors. To implement (3.17), we can use two AND operations for a_{k-1}^b and w_k^b as well as their complements. Therefore, the bitwise convolution on RRAM-crossbar can be shown as follows:

$$s_k = \sum_{i=1}^N w_k^b \otimes a_{k-1}^b = \sum_{i=1}^N (w_k^b \cdot a_{k-1}^b + \overline{w_k^b} \cdot \overline{a_{k-1}^b}) \quad (5.5)$$

The mapping of the bitwise convolution is shown in Figure 5.3(a). It requires a $2N \times N$ RRAM-crossbar, where N is the number of element in the vector. All columns are configured with the same elements that correspond to one column in binary weight w_k^b of the neural network, and the WL voltages are determined by the binary input a_{k-1}^b . Due to the large ratio between R_{off} and R_{on} , the current through the BL is approximately equal to $\frac{s_k V_r R_s}{R_{on}}$, where s_k is the inner-product result in (3.17). Since the current of all BLs is identical, the ladder-like threshold voltages $V_{th,j}$ are set as follows:

$$V_{th,j} = \frac{(2j+1)V_r R_s}{2R_{on}}, \quad (5.6)$$

where $V_{th,j}$ is the threshold voltage for the j_{th} column. If we use s' to denote the output array, and $s'(j)$ to denote the output of column j in RRAM-crossbar, we can have

$$s'_k(j) = \begin{cases} 1, & j < s_k \\ 0, & j \geq s_k \end{cases} \quad (5.7)$$

In this case, the inner-product results s_k can be recognized that the first $(N - s_k)$ output bits are 0 and the rest s_k bits are 1. The relation between s'_k and s_k can be expressed as $s'_k = g(s_k)$.

As described in (3.17), each binary weight vector w_k performs bitwise convolution with several input features. As a result, each logic block in Figure 4.10 stores a binary vector w_k , while the control bus transmits the input feature sequentially. In this case, bitwise convolution can be performed in parallel in separated logic blocks.

Figure 5.3: Digital RRAM-crossbar mapping for (a) bitwise convolution, (b) bitwise batch normalization, (c) bitwise max-pooling.

5.1.2.2 Mapping Bitwise Batch Normalization

Bitwise batch normalization requires two digital RRAM-crossbars in the implementation. In the first RRAM-crossbar, it performs the XOR operation on adjacent bits of the output of bitwise convolution. It can be expressed as

$$s''_k(j) = \begin{cases} 1, & j = s_k \\ 0, & j \neq s_k \end{cases} \quad (5.8)$$

After that, the second RRAM-crossbar builds a look-up-table (LUT). Since μ , σ^2 , γ and β are all fixed in the inference stage, (3.18) can be rewritten as

$$a_k = f(s_k), \quad (5.9)$$

where $f(\cdot)$ represents the LUT. As a result, the LUT is stored in the second RRAM-crossbar according to the parameters μ , σ^2 , γ and β . As described in (5.8), only the s_k -th row of the LUT is selected, so the batch normalization result can be directly readout. The threshold voltage of both two RRAM-crossbars are

$$V_{th} = \frac{V_r R_s}{2R_{on}} \quad (5.10)$$

To have a better illustration, Figure 5.3(b) shows the detailed mapping and Table 5.1 shows the values to store in the second RRAM-crossbar when $\mu = 2.5$, $\sigma^2 = 5$, $\gamma = 1$ and $\beta = 0$ referred to the IEEE-754 standard.

Table 5.1: Binary format stored in RRAM-crossbar

BinConv output	Batch normalization	Floating binary format
0	-0.5	10111111000000000000000000000000
1	-0.3	10111110100110011001100110011010
...	...	
7	0.9	00111111011001100110011001100110

5.1.2.3 Mapping Bitwise Pooling and Binarization

According to Equation (3.21), we will do the binarization first and then perform max-pooling. The bitwise activation in Equation (3.20) can be achieved by selecting the sign-bit of the binary format output of Figure 5.3(b). In max-pooling, the output is 0 only when all the numbers in the pooling region are negative. As a result, we can add all the complementary of sign-bit in the pooling region. If the result is not 0, it indicates that at least one positive number is in the pooling region, resulting in the pooling result 1. In summary, the max-pooling in (3.19) can be re-written as

$$a'_k(i, j) = \begin{cases} 1, & \text{if } \sum \overline{\text{Sign}\{a_k(Mi + k, Mj + l)\}} > 0 \\ 0, & \text{otherwise} \end{cases} \quad (5.11)$$

As a result, both of the bitwise max-pooling and binarization can be implemented by the addition operation performed on the digital RRAM-crossbar with 1 column, as shown in Figure 5.3(c).

5.1.2.4 Summary of Mapping Bitwise CNN

As a summary, the four operations of the bitwise CNN in Figure 3.7(b) can be fully mapped onto the digital RRAM-crossbar. All the threshold voltages are fixed even if the parameters of bitwise CNN are changed. Although these operations are implemented in different RRAM-crossbar array, the input/output formats of them are compatible so that one can directly connect them as shown in the logic block in Figure 4.10 with pipeline used. The pipeline design is based on Table 5.2, so that each stage implements a layer. CONV-2, CONV-4 and CONV-6 are the stages which require the most RRAM cells and computation time.

As a result, more logic blocks are assigned to these steps to relieve the critical path. In our simulation, half of the digitized RRAM-crossbars are used to perform these three layers. Moreover, since the output feature from Figure 5.3(c) is binary, the area overhead and energy consumption of the data storage can be significantly reduced. In addition,

Table 5.2: Bitwise CNN configuration of CIFAR-10 dataset.

Name	Filter/weight	Output Size
CONV-1	$3 \times 3 \times 3 \times 128$	$128 \times 32 \times 32$
CONV-2	$3 \times 3 \times 128 \times 128$	$128 \times 32 \times 32$
MP-2	2×2	$128 \times 16 \times 16$
CONV-3	$3 \times 3 \times 128 \times 256$	$256 \times 16 \times 16$
CONV-4	$3 \times 3 \times 256 \times 256$	$256 \times 16 \times 16$
MP-4	2×2	$256 \times 8 \times 8$
CONV-5	$3 \times 3 \times 256 \times 512$	$512 \times 8 \times 8$
CONV-6	$3 \times 3 \times 512 \times 512$	$512 \times 8 \times 8$
MP-6	2×2	$512 \times 4 \times 4$
FC-1	8192×1024	1024
FC-2	1024×1024	1024
FC-3	1024×10	10

because layers in Table 5.2 are implemented in different data-logic pairs, the volume of data transmitted is also decreased.

5.1.3 BCNN Based Inference Acceleration on 1S1R Array

In this work, we will present mappings of bitwise convolutional neural network (BCNN) operations on the sneak-path-free digital 1S1R RRAM-crossbar. The mapping detail is a little different compared to the mapping on passive array. Different from the signed realization in previous works [91, 47], we will use unsigned bitwise operation in the mapping.

5.1.3.1 Mapping Unsigned Bitwise Convolution

As described in Section 3.2.1, dot-product operations is the majority in bitwise convolution. Suppose that a single dot-product operation has N numbers, the basic operation can be expressed as

$$s_{x,y} = \sum_{i=1}^N a_{x,i} \cdot w_{i,y} \quad (5.12)$$

where $a_{x,i}, w_{i,y} \in \{-1, +1\}$. To implement the negative weights, [91, 47] use two RRAM-crossbars to represent positive and negative weights, respectively. However, this solution requires a subtract operation for every complementary BL output. It is obvious that the subtract implementation leads to extra CMOS circuit with more area and energy-consumption. To avoid this problem, one can apply the conversion by XNOR operation ($\odot$) in $\{0, 1\}$ introduced in [92, 67], as shown in Table 5.3.

Figure 5.4: Mapping of bitwise CNN on sneak-path-free digital RRAM-crossbar

Since $a' \odot w' = a' \cdot w' + \bar{a}' \cdot \bar{w}'$, we can directly map $a', \bar{a}'$ as WL inputs, and $w', \bar{w}'$ as

Table 5.3: Conversion from signed bitwise convolution to unsigned bitwise XNOR

a	w	Multiplication	a'	w'	XNOR
+1	+1	+1	1	1	1
+1	-1	-1	1	0	0
-1	+1	-1	0	1	0
-1	-1	+1	0	0	1

RRAM conductances. We can use s' to denote the new bitwise convolution result as

$$s'_{x,y} = \sum_{i=1}^N a'_{x,i} \odot w'_{i,y} = \sum_{i=1}^N (a'_{x,i} \cdot w'_{i,y} + \overline{a'_{x,i}} \cdot \overline{w'_{i,y}}) \quad (5.13)$$

Since s must be an even integer and $s \in [-N, N]$ while $s' \in [0, N]$, the relation can be summarized as $s = 2s' - N$. Although this mapping strategy needs the same number of RRAM devices as the one in [91, 47], it only needs an inverter for each input bit.

5.1.3.2 Mapping Batch Normalization, Pooling and Binarization

Section 3.3.3 has explained these three steps with detailed equations, but it is simpler to map them on RRAM-crossbar. We can notice that the order of binarization and pooling does not affect the final output. Therefore, by combining batch normalization and binarization, we can use $c = \mu - \beta\sqrt{\sigma^2}/\gamma$ from BatchNorm (parameters for the $\{0, 1\}$ convolution fashion) as the threshold to compare with the bitwise convolution result. Although the corresponding voltage of c is generated by limited-bit DAC, it is still more energy-efficient than the ADC for analog RRAM-crossbar. The comparison process is binarization with only 0 or 1 output. These two steps can be summarized as

$$Out = \begin{cases} 1, & \sum_{i=1}^N a_{x,i} \odot w_{i,y} \geq \mu - \beta\sqrt{\sigma^2}/\gamma \\ 0, & \sum_{i=1}^N a_{x,i} \odot w_{i,y} < \mu - \beta\sqrt{\sigma^2}/\gamma \end{cases} \quad (5.14)$$

As for pooling operation, we can put a single-output RRAM-crossbar and configure the comparator threshold as $\frac{V_r R_s}{2(R_s + R_{LRS})}$ to detect if any binarization result in the pooling region is 1.

Figure 5.4 shows the summary of bitwise CNN mapping on the proposed digital RRAM-crossbar. Since the output of each BL is also in $\{0, 1\}$ fashion, it can be directly used for the binary input feature of the next layer.

5.1.4 L2-Norm Gradient Based Learning and Inference Acceleration

The details of this section has been presented in [93]. In this work, we will map the multi-layer feed-forward neural network based learning and inference processes according to the algorithm and network in Section 3.2.2. We will focus on the mapping of all the matrix-vector multiplication operations and L2-norm calculation on the passive array and coupled oscillator based XIMA.

5.1.4.1 Mapping Matrix-Vector Multiplication on RRAM-Crossbar Network

With the introduction of binary-memristor-crossbar network, one can perform matrix-vector multiplication [45]. It has much better non-uniformity under process variation when compared to the multiple-level valued analog memristor crossbar. As discussed in Section 3.2.2, the key operations in machine learning (feature extraction and convolution) requires a large number of matrix-vector multiplication.

Here, we take the operation of $\mathbf{Y} = \mathbf{H}\mathbf{W}$ as an example for mapping the multiplication on the crossbar. For every element in $\mathbf{Y}$, it follows

$$y_{ij} = \sum_{k=1}^N h_{ik} w_{kj}, \quad (5.15)$$

where h and w are the elements in $\mathbf{H}$ and $\mathbf{W}$, respectively.

For the crossbar structure in Figure 4.10, the current through each BL follows

$$I_j = \sum_{i=1}^m \frac{V_{wl,i} R_{fixed}}{R_{ij}} \quad (5.16)$$

As a result, we can map h_{ik} and w_{kj} on $V_{wl,i}$ and R_{ij} , respectively. By measuring the current through BL with binary digitization, we can derive the result y_{ij} in the digital domain. The detailed operation can be referred to the work in [45].

Note that for real numbers, we only need a few more shifted addition to complete the final process of matrix-vector multiplication with memristor crossbar. Figure 5.5 shows that we only need 4-shifted addition with 8-bit precision to calculate the final inner product. In addition, Sigmoid function can be implemented in memristor crossbar with look-up table (LUT).

5.1.4.2 Mapping L2-Norm Calculation on Coupled RRAM Oscillator Network

In a coupled oscillator network, we use the memristor oscillator cell similar with Figure 2.15(a) (Rs replaced with a PMOS). By altering the gate voltage of the PMOS, we can control the oscillatory frequency as in Figure 2.15(d). Referring to previous works [84, 85], the proposed coupled-memristor-oscillator network is shown in Figure 4.8(a). Here, oscillators are directly coupled with memristors. The oscillatory frequency of each oscillator is determined by multiple input voltages. As mentioned in Figure 2.15(d), the voltage-controlled PMOS can be viewed as a voltage-controlled resistor. Figure 4.8(b) and Figure 4.8(c) are the schematic of oscillators with one input and multiple inputs. After directly coupling, we have an inverter and a feedback resistor with a diode-connected NMOS. These elements guarantee that the current will only pass from the oscillator net-

Figure 5.5: (a) Crossbar network for multiplication; (b) shifted additions

work to V_{out} . A RC circuit is design at V_{out} so that we can measure the DC voltage of V_{out} . The output voltage indicates the relative synchronization of the coupled oscillators.

We can build a mapping function as follows for the oscillator condition:

$$\begin{aligned} V_{ik1} &= V_{bias} + \alpha t_{ik} \\ V_{ik2} &= V_{bias} - \alpha y_{ik}, \end{aligned} \quad (5.17)$$

where α is a scale factor to link the oscillator input voltage; and t_{ik} and y_{ik} are the elements in matrices $\mathbf{T}$ and $\mathbf{Y}$, respectively.

As mentioned in [84, 85], the coupled oscillator network can work as a degree-of-match (DOM) circuit to calculate L2-norm by fitting V_{out} to 2^{nd} polynomial when there are two inputs for each oscillator in Figure 4.8(c). Therefore, the output voltage follows

$$V_{out} = V_{min} + \eta ||\mathbf{T} - \mathbf{Y}||_2^2 = V_{min} + \eta \sum_{k=1}^M \sum_{i=1}^N (t_{ik} - y_{ik})^2, \quad (5.18)$$

where η is a fitting parameter, and V_{min} is the output voltage when $||\mathbf{T} - \mathbf{Y}||_2^2 = 0$.

As such, we can use the coupled-memristor-oscillator network to the L2-norm error function in machine learning. The L2-norm result is obtained by curve fitting but not accurate computing. Compared to other implementations, such an oscillator network

Figure 5.6: Mapping flow of machine learning on memristor network.

can complete the operation much faster because the square operation $(t_{ik} - y_{ik})^2$ can be performed simultaneously by the oscillator circuit. Moreover, the accumulation is also automatically achieved by the coupled network.

5.1.4.3 Mapping Flow of Multi-Layer Neural Network on RRAM Network

Based on the coupled-memristor-oscillator network and binary-memristor-crossbar network, we can accelerate direct-gradient based L2-norm minimization for machine learning as shown in Figure 5.6. For the model part, we use two types: memristor oscillator and memristor crossbar. We find the parameter in oscillator network to calculate L2-norm; and map on crossbar network to calculate digitized matrix-vector multiplication. Afterwards, we generate the initial weights of A and W to calculate the gradient δE . The best direction to minimize E can be derived. This process will be repeated for several times till the minimization is reached only when $E^k - E^{k-1} < \text{threshold}$.

5.2 Machine Learning Algorithms on 3D Multi-Layer CMOS-RRAM Architecture

We will show the detailed mapping of several machine learning applications on 3D CMOS-RRAM architecture, including SLFN based learning and TNN inference.

5.2.1 On-Chip Design for SLFN

The details of this section has been presented in [94]. In this work, we will map the single-layer feed-forward neural network based learning and inference processes according to the algorithm and network in Section 3.2. We will focus on the mapping of all the matrix-vector multiplication operations on RRAM-crossbar and other operations in CMOS. Multi-layer 3D CMOS-RRAM architecture in Figure 4.13 is used for the acceleration.

Here, we will firstly discuss the data quantization on the proposed architecture. Then we will illustrate how to map matrix-vector multiplication on RRAM-crossbar layer. Finally, CMOS-ASIC accelerator is also designed in layer 3 for remaining operations in pipelined and parallel fashion.

5.2.1.1 Data Quantization

To implement the whole training algorithm on the proposed 3D multi-layer CMOS-RRAM accelerator, the precision of the real values requires a careful evaluation. Compared to the software double precision floating point format (64-bit), the values in the training algorithm are truncated into finite precision. A general N_b bit fixed point representation with m_b bit integer and n_b bit fraction point is shown as below.

$$y = -b_{N_b-1}2^{m_b} + \sum_{l=0}^{N_b-2} b_l 2^{l-n_b} \quad (5.19)$$

In the proposed accelerator, we first assign a fixed 16 bit-width for normalized input data such as $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{D}$ with scale factor $m_b = 0$ and $n_b = 16$. For weights such as $\mathbf{A}$ and $\mathbf{B}$ in SLFN, we determine m_b by finding the logarithm of dynamic range (i.e. $\log_2(\text{Max} - \text{Min})$) and n_b is set as $16 - m_b$. Furthermore, we applied greedy search method based on the testing accuracy to find the optimal bit width. The bit width is reduced with cross-validation applied to evaluate the effect. By dynamic tuning the bit width, our objective is to find the minimum bit width with acceptable testing accuracy loss.

Figure 5.7: Detailed mapping for digitalized matrix-vector multiplication on RRAM-crossbar: (a) Parallel digitizing (b) XOR (c) Encoding and (d) Legend of mapping

Figure 5.8: Inner-product operation on RRAM-crossbar

5.2.1.2 RRAM Layer Implementation for Digitized Matrix-Vector Multiplication

The matrix-vector multiplication is illustrated with the example of output layer (3.7) calculation, where each element in $\mathbf{Y}$ is calculated using $y_{ij} = \sum_{k=1}^L h_{ik} \gamma_{kj}$. Such multiplication can be expressed in binary multiplication [45].

$$\begin{aligned}
 y_{ij} &= \sum_{k=1}^L \left(\sum_{e=0}^{E-1} B_e^{h_{ik}} 2^e \right) \left(\sum_{g=0}^{G-1} B_g^{\gamma_{kj}} 2^g \right), \\
 &= \sum_{e=0}^{E-1} \sum_{g=0}^{G-1} \left(\sum_{k=1}^L B_e^{h_{ik}} B_g^{\gamma_{kj}} 2^{e+g} \right) = \sum_{e=0}^{E-1} \sum_{g=0}^{G-1} s_{eg} 2^{e+g}
 \end{aligned} \tag{5.20}$$

where s_{eg} is the accelerated result from RRAM-crossbar. $B_e^{h_{ik}}$ is the binary bit of h_{ik} with E bit-width and $B_g^{\gamma_{kj}}$ is the binary bit of γ_{kj} with G bit-width. As mentioned above, bit-width E and G are decided using cross validation in design to achieve balanced accuracy and hardware usage [95].

Step 1: Parallel Digitizing: It requires an $L \times L$ RRAM-crossbar. Each inner-

Figure 5.9: Detailed pipelined and parallel mapping of CMOS-ASIC accelerator

product is produced RRAM-crossbar as shown in Figure 5.7(a). h_i is set as crossbar input and γ is written in RRAM cells. In the $L \times L$ RRAM-crossbar, resistance of RRAMs in each column are the same, but V_{th} among columns are different. As a result, the output of each column mainly depends on ladder-like threshold voltages $V_{th,j}$. If the inner-product result is s , the output of step 1 is like $(1 \dots 1, 0 \dots 0)$, where $O_{1,s} = 1$ and $O_{1,s+1} = 0$.

Step 2: XOR: It is to identify the index of s with the operation $O_{1,s} \oplus O_{1,s+1}$. Note that $O_{1,s} \oplus O_{1,s+1} = 1$ only when $O_{1,j} = 1$ and $O_{1,j+1} = 0$ from Step 1. The mapping of RRAM-crossbar input and resistance is shown in Figure 5.7(b), and threshold voltage configuration is $V_{th,j} = \frac{V_r R_s}{2R_{on}}$. Therefore, the index of s is identified by XOR operation.

Step 3: Encoding: the third step produces s in binary format as an encoder with the thresholds as Step 2. The input from the second step produces $(0 \dots 1, 0 \dots 0)$ like result where only the s^{th} input is 1. As a result, only the s^{th} row is read out and no current merge occurs in this step. The according binary format $binary(s)$ is an intermediate result and stored in the s^{th} row, as shown in Figure 5.7(c). Encoding step needs an $L \times n$ RRAM-crossbar, where $n = \lceil \log_2 L \rceil$ is the number of bits in order to represent 1 to L in binary format.

5.2.1.3 CMOS Layer Implementation for Decoding and Incremental Least-Squares

In the CMOS layer, decoding and more complex operations for incremental least-squares solution are designed. Since the output from RRAM layer is in binary format, decoding is required to obtain the real values. Adder and shifter are designed in the layer 3 with CMOS to complete this process as shown in Figure 5.8.

To fully map the incremental machine learning on the proposed 3D multi-layer CMOS-RRAM accelerator, there are following operations needed in CMOS-ASIC including: sorting in Equation (3.3), non-linear mapping in Equation (3.4) and division. Figure 5.9 shows the detailed mapping of the supervised classification on the proposed 3D multi-layer CMOS-RRAM accelerator. The RRAM logic layer (layer 2) will work as vector core with parallel process elements (PE) for multiplication and summation. The scalar core is implemented to perform scalar operation including division. A sequential divider is implemented with 5-stage pipelines to reduce critical path latency. Non-linear mapping such as Sigmoid function for activation is also implemented in look-up table (LUT) (3.4). As a result, the whole training process including both feature extraction and classifier learning can be mapped to the proposed 3D multi-layer CMOS-RRAM accelerator with small accuracy loss.

5.2.2 On-Chip Design for TNN

5.2.2.1 Mapping on Multi-Layer Architecture

In this work, we will map a two-layer tensorized neural network (TNN) based inference processes according to the algorithm and network in Section 3.4. we further discuss how to utilize the proposed 3D multi-layer CMOS-RRAM architecture to design the TNN accelerator. We first discuss the CMOS layer design, which performs the high level control of TNN computation. Then a highly-parallel RRAM based accelerator is introduced with the TNN architecture and dot-product engine.

CMOS Layer Accelerator To fully map TNN on the proposed 3D multi-layer CMOS-RRAM accelerator, the CMOS logic is designed mainly for logic control and synchronization using top-level state machine. It prepares the input data for computing cores, monitors the states of RRAM logic computation and determines the computation layer of neural network. Figure 5.10 shows the detailed mapping of the tensorized neural network (TNN) on the proposed 3D multi-layer CMOS-RRAM accelerator. This is a folded architecture by utilizing the sequential operation of each layer on the neural network. The testing data will be collected from RRAM memory through TSV and then sent into

addition, subtraction and comparisons. Then the output from the scalar core will be sent to the sigmoid function model for activation matrix in a pipelined fashion, which performs the computation of (3.4). The activation matrix $\mathbf{H}$ will be used for the next layer computation. As a result, the whole TNN inference process can be mapped to the proposed 3D multi-layer CMOS-RRAM accelerator.

In addition, to support TNN on RRAM computation, a dedicated index look-up table is formed. Since the weight matrix is actually folded into a high dimensional tensor as shown in Figure 3.8, a correct index selection function called bijective function is designed. The bijective function for weight matrix index is also performed by the CMOS layer. Based on the top state diagram, it will choose the correct slice of tensor core $\mathbf{G}_i[i, j]$ by determining the i, j index. Then the correct RRAM-crossbar area will be activated to perform vector-matrix multiplication.

RRAM Layer Accelerator In the RRAM layer, we design the RRAM layer accelerator for highly-parallel computation using single instruction multiple data (SIMD) method to support data parallelism.

Highly-parallel TNN Accelerator on the RRAM Layer The TNN accelerator is designed to support highly parallel tensor-train-matrix-by-vector multiplication by utilizing the associative principle of matrix product. According to Equation (3.27), $\mathcal{X}(\mathbf{i})$ needs to be multiplied by d matrices unlike the general neural network. As a result, if traditional matrix-vector multiplication in serial is applied, data needs to be stored in the RRAM array for d times, which is time-consuming. Since the size of tensor cores in the TNN is much smaller than the weights in the general neural network, multiple matrix-vector multiplication engines can be placed in the RRAM logic layer. When then input data is loaded, the index of $\mathbf{G}_i$ can be known. For example, we need compute $\mathbf{X}(\mathbf{j})\mathbf{G}_1[i_1, j_1]\mathbf{G}_2[i_2, j_1]\mathbf{G}_3[i_3, j_1]\mathbf{G}_i[i_4, j_1]$ given $d = 4$ for the summation in (3.27). $\mathbf{G}_1[i_1, j_1]\mathbf{G}_2[i_2, j_1]$ and $\mathbf{G}_3[i_3, j_1]\mathbf{G}_i[i_4, j_1]$ in (3.27) can be pre-computed in a parallel fashion before the input data $\mathcal{X}(\mathbf{i})$ is loaded.

As shown in Figure 5.11, the tensor cores (TC1-6) are stored in the RRAM logic layer. When the input data $\mathcal{X}(\mathbf{j})$ comes, the index of each tensor core is loaded by the logic layer controllers first. The controller will write the according data from the tensor cores to RRAM cells. As a result, the matrix-vector multiplication of $\mathbf{G}_i$ can be performed in parallel to calculate the intermediate matrices while $\mathcal{X}(\mathbf{i})$ is in the loading process. After all the intermediate matrices are obtain, they can be multiplied by $\mathcal{X}(\mathbf{i})$ so that the operations in RRAM logic layer can be efficient.

Figure 5.11: (a) RRAM based TNN accelerator (b) tree-based parallel partial result multiplication (c) RRAM based dot-product engine

Highly-Parallel Dot-product Engine on The RRAM Layer We further develop the digitalized RRAM based dot-product engine on the RRAM layer. The tensor-train-matrix-by-vector operation can be efficiently accelerated by the fast dot-product engine on the RRAM layer, as shown in Figure 5.11(b). By taking correct index of cores, each operation can be divided into a vector-vector dot product operation. Here, we design the dot-product engine based on [96]. We use the output matrix Y , input matrices X and Φ for better understanding. The overall equation is $Y = X\Phi$ with $Y \in \mathbb{R}^{M \times m}$, $X \in \mathbb{R}^{M \times N}$

and $\Phi \in \mathbb{R}^{N \times m}$. For every element in $\mathbf{Y}$, it follows

$$y_{ij} = \sum_{k=1}^N x_{ik} \phi_{kj}, \quad (5.21)$$

where x and ϕ are the elements in $\mathbf{X}$ and Φ respectively. The basic idea of implementation is to split the matrix-vector multiplication to multiple inner-product operations of two vectors x_i and ϕ_j . Such multiplication can be expressed in binary multiplication by adopting fixed point representation of x_{ik} and ϕ_{kj} :

$$\begin{aligned} y_{ij} &= \sum_{k=1}^N \left(\sum_{e=0}^{E-1} B_e^{h_{ik}} 2^e \right) \left(\sum_{g=0}^{G-1} B_g^{\gamma_{kj}} 2^g \right), \\ &= \sum_{e=0}^{E-1} \sum_{g=0}^{G-1} \left(\sum_{k=1}^N B_e^{h_{ik}} B_g^{\gamma_{kj}} 2^{e+g} \right) = \sum_{e=0}^{E-1} \sum_{g=0}^{G-1} s_{eg} 2^{e+g} \end{aligned} \quad (5.22)$$

where s_{eg} is the accelerated result from RRAM-crossbar. $B^{h_{ik}}$ is the binary bit of h_{ik} with E bit-width and $B^{\gamma_{kj}}$ is the binary bit of γ_{kj} with G bit-width. As mentioned above, bit-width E and G are decided during the algorithm level optimization. The matrix-vector multiplication based on (5.22) can be summarized in four steps on the RRAM layer.

Step 1: Index bijection: Select the correct slice of tensor cores $\mathbf{G}_d[i_d, j_d] \in \mathbb{R}^{r_d \times r_{d+1}}$, where a pair of $[i_d, j_d]$ determines a slice from $\mathbf{G}_d \in \mathbb{R}^{r_d \times n_d \times r_{d+1}}$. In our current example, we use $\mathbf{X}$ and Φ to represent two selected slices from cores $\mathbf{G}_1$ and $\mathbf{G}_2$.

Step 2: Parallel Digitizing: It requires an $N \times N$ RRAM-crossbar. Each inner-product is produced RRAM-crossbar as shown in Figure 5.11(c). h_i is set as crossbar input and γ is written in RRAM cells. In the $N \times N$ RRAM-crossbar, resistance of RRAMs in each column are the same, but V_{th} among columns are different. As a result, the output of each column mainly depends on ladder-like threshold voltages $V_{th,j}$. If the inner-product result is s , the output of step 1 is like $(1...1, 0...0)$, where $O_{1,s} = 1$ and $O_{1,s+1} = 0$.

Step 3: XOR: It is to identify the index of s with the operation $O_{1,s} \oplus O_{1,s+1}$. Note that $O_{1,s} \oplus O_{1,s+1} = 1$ only when $O_{1,j} = 1$ and $O_{1,j+1} = 0$ from Step 1. The mapping of RRAM-crossbar input and resistance is also shown in 5.11(c), and threshold voltage configuration is $V_{th,j} = \frac{V_T R_s}{2R_{on}}$. Therefore, the index of s is identified by XOR operation.

Step 4: Encoding: the third step produces s in binary format as an encoder with the thresholds as Step 2. The input from the second step produces $(0...1, 0...0)$ like result where only the s^{th} input is 1. As a result, only the s^{th} row is read out and no current merge occurs in this step. The according binary format $binary(s)$ is an intermediate result and stored in the s^{th} row, as shown in Figure 5.11(c). Encoding step needs an

$N \times n$ RRAM-crossbar, where $n = \lceil \log_2 N \rceil$ is the number of bits in order to represent 1 to N in binary format.

By applying these four steps, we can map different tensor cores on RRAM-crossbars to perform the matrix-vector multiplication in parallel. Compared to the state-of-arts realizations, this approach can perform the matrix-vector multiplication faster and more energy-efficient.

5.2.2.2 Mapping on Single-Layer Architecture

In this work, we will map a two-layer tensorized neural network (TNN) based inference processes according to the algorithm and network in Section 3.4. we further discuss how to utilize the proposed 3D single-layer CMOS-RRAM architecture to design the TNN accelerator. Compared to the mapping in Section 5.2.2, we achieve a higher parallelism with a distributed mapping.

TNN Accelerator Design According to Equation (3.27), $\mathcal{X}(\mathbf{i})$ needs to be multiplied by d matrices unlike the general neural network. In this equation, we only need to take a slice of 3-dimensional matrix (i.e., a 2-d matrix) from each tensor core to perform the matrix-vector multiplication. Therefore, the computation complexity can be significantly reduced. We show the detailed accelerator design between two hidden layers of TNN in Figure 5.12. Since the parameters of the neural network are compressed, we store all the weights of TNN in RRAM-crossbar, following Figure 4.12(b). The input data $\mathbf{H}_1$ is processed by several tensor cores, CMOS pooling and activation serially.

The detailed design of a tensor core is also shown in Figure 5.12. In each tensor core, we store different slices of the 3-dimensional matrix into different RRAM-crossbars. Since only one 2D matrix is used at a time, two tensor core Multiplexers (MUX) are used so that only one matrix is connected to the input voltage as well as the output ADC. The TC selection module controls the input and output MUX according to i and j . For the tensor cores in the middle, the ADC/DAC pair can be removed since we don't need a digital representation for RRAM-crossbar inputs.

Benefits Summary of TNN on 3D CMOS-RRAM Architecture From the algorithm optimization perspective, such tensorization can benefit of implementing large neural networks. Firstly, by performing tensorization, the size of neural network can be compressed. Moreover, the computation load can also be reduced by adopting small tensor ranks. Secondly, a tensorization of weight matrix can decompose the big matrix into many small tensor-core matrices, which can effectively reduce the configuration time of

Figure 5.12: Mapping details of highly-parallel 3D single-layer CMOS-RRAM accelerator for TNN

RRAM. Lastly, the multiplication of small matrix can be performed in a highly parallel fashion on RRAM to speed-up the large neural network processing time.

From the hardware computing architecture perspective, the proposed 3D CMOS-RRAM accelerator can greatly improve the energy efficiency and neural network processing speed. The matrix-vector multiplication can be intrinsically implemented by RRAM-crossbar. Compared to CMOS, a multi-bit tensor core weight can be represented by a single RRAM, and the addition can be realized by current merging. Moreover, as all the tensor cores for the neural network weights are stored in the RRAM devices, better area-efficiency can be achieved. The power consumption of the proposed CMOS-RRAM is much smaller than the CMOS implementation due to non-volatile property of RRAM. In the TNN accelerator design shown in Figure 5.12, only a slice of the

3-dimensional matrix is active while most of the matrices are inactive. For traditional CMOS realization, it costs huge static power for the inactive tensor core weights. But in the proposed architecture, the inactive matrices do not consume power so that the energy utilization is high, resulting a better energy-efficiency.

Chapter 6

Results

In this chapter, we will show mapping results of all the applications in Section 5. We will introduce the detailed mapping strategy, experimental settings and evaluation results.

6.1 Results on XIMA

We will show the detailed mapping of several machine learning applications on XIMA, including SLFN based learning, BCNN based inferences on passive array as well as 1S1R array.

6.1.1 Results of SLFN

6.1.1.1 SLFN Based Learning and Inference Acceleration

Experiment Settings The hardware evaluation platform is implemented on a computer server with 4.0GHz core and 16.0GB memory. Feature extraction is implemented by general processor, CMOS-based ASIC, non-distributed and distributed in-memory computing based on digitalized RRAM-crossbar respectively. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as $1k\Omega$ and $1M\Omega$ as on-state and off-state resistance respectively according to [81].

The general processor solution is performed by Matlab on a 4.0GHz desktop. For CMOS-ASIC implementation, we implement it by Verilog and synthesize with CMOS 65nm low power PDK. For RRAM-crossbar based solution, we verify the function in circuit level with SPICE tool NVMSPIICE [25]. By analyzing the machine learning algorithm, we obtain the basic operations and the number of RRAM-crossbar logic units required.

The working frequency of general processor implementation is 4.0GHz while the

Table 6.1: Face recognition performance comparison under among software and hardware implementations.

Implementation	Frequency	Area	Power	Type	Time	Energy
CPU (Matlab)	4.0GHz	$177mm^2$	84W	PCA	1.56ms	131mJ
				Input layer	0.98ms	82mJ
				L2-norm	92.5ms	7.77J
				Output layer	0.1ms	8.40mJ
				Overall	95.14ms	7.99J
CMOS- ASIC	1.0GHz	$5.64mm^2$	39.41W	PCA	195 μs	7.68mJ
				Input layer	121.9 μs	4.80mJ
				L2-norm	12.26ms	483mJ
				Output layer	12.4 μs	0.49mJ
				Overall	12.59ms	496mJ
				Improvement	7.56 $\times$	16.11 $\times$
Distributed in-memory RRAM- crossbar architecture	200MHz	$0.11mm^2$	13.1W	PCA	38.4 μs	0.5mJ
				Input layer	24 μs	0.3mJ
				L2-norm	2.83ms	37.1mJ
				Output layer	2.4 μs	30 μJ
				Overall	2.90ms	37.93mJ
				Improvement	32.84 $\times$	210.69 $\times$

CMOS ASIC feature extraction design frequency is 1.0GHz. For in-memory computing based on the proposed RRAM-crossbar, write voltage V_w is set as 0.8V and read voltage V_r is set as 0.1V as well as duration time of 5ns. In addition, the analog computation on RRAM-crossbar is performed for comparison based on design in [82].

In the followings, we will show the performance of matrix-vector multiplication on RRAM-crossbar first. A scalability study is introduced to show the area, energy and computation delay with different matrix sizes. Afterwards, the evaluation of face recognition on in-memory architecture is presented. Finally, we will illustrate the object classification on 3D CMOS-RRAM architecture. Performance of different bit-width configurations will also be shown. In addition, the 3D CMOS-RRAM solution will be compared with CMOS-ASIC as well as GPU implementation.

Performance Evaluation In this work, we implement the face recognition application on the in-memory architecture. We will analyze the computation complexity of face recognition first, and then evaluate the performance.

In the experiment, 200 face images of 13 people are selected from [97], with scaled image size 262 of each image $\mathbf{X}$. In PCA, feature size of image is further reduced to 128 by multiplying the matrix $\mathbf{R}$. The number of hidden node L and classes m are

Figure 6.1: (a) Time consumption breakdown for output weight calculation (b) neural network training computation effort analysis (c) Multiplication analysis for neural network training ($N = 200$, $n = 128$, $L = 160$ and $m = 13$)

160 and 13, respectively. Based on the experimental settings, computation complexity is analyzed with results shown in Figure 6.1. 82% of computations are multiplication in output weight calculation, which is the most time-consuming procedure in neural network. Time-consumption of each process in neural network is introduced in Figure 6.1(b). Since processes except activation function involve matrix-vector multiplication, we extracted this operation in the whole algorithm and found that 64.62% of time is consumed in matrix-vector multiplication, shown in Figure 6.1(c).

We implement the face recognition applications by the SLFN on XIMA. In Table 6.1, general performance comparisons among Matlab, CMOS-ASIC and RRAM-crossbar accelerator are introduced, and the acceleration of each procedure as the formula described in Section 3.2.1 is also addressed. Among three implementations, RRAM-crossbar architecture performs the best in area, energy and speed. Compared to Matlab implementation, it achieves $32.84\times$ speed-up, $210.69\times$ energy-saving and almost four-magnitude area-saving. We also design a CMOS-ASIC implementation with similar structure as RRAM-crossbar with better performance compared to Matlab. RRAM-crossbar architecture is $4.34\times$ speed-up, $13.08\times$ energy-saving and $51.3\times$ area-saving compared to CMOS-ASIC. The performance comparison is quite different from Table 4.1, because we applied different designs (RRAM-crossbar size) in these two experiments according to the dimension of matrices.

The result of face recognition is shown in Figure 6.2. Five training classes are provided as an example with three test cases. Each test face will be recognized as the class with the largest score (prediction result as the index of $Max(\mathbf{Y})$, marked in red color). In this example, case 1 is identified as class 1, while case 2 and 3 are classified into class 3 and 5, respectively.

Class 1		Test cases/ classes	Test case 1 (Y)	Test case 2 (Y)	Test case 3 (Y)
Class 2		Class 1	1.518062	-0.79108	-0.58029
Class 3		Class 2	-0.29803	-0.87155	-0.24397
Class 4		Class 3	-0.95114	0.793256	-0.35867
Class 5		Class 4	-0.65597	-0.44714	-0.70879
		Class 5	-0.65955	0.262689	0.872497
		•	•	•	•
		•	•	•	•
		•	•	•	•

Figure 6.2: Training samples and prediction value $\mathbf{Y}$ from Equation (3.7) for face recognitions.

6.1.1.2 L2-Norm Gradient Based Learning and Inference Acceleration

Experiment Settings The hardware evaluation platform is implemented on a computer server with 4.0GHz core and 16.0GB RAM. For RRAM based oscillator network, we use the VerilogA model from [51] and device model from [49] as the RRAM before forming process and simulate the network in Cadence Spectre. For RRAM based crossbar network, the resistance of RRAM is set as $1k\Omega$ and $1M\Omega$ as on-state and off-state resistance respectively according to [81]. NVM-SPICE [25] is used for RRAM-crossbar network verification. For comparison propose, we also implement the L2-norm-gradient based machine learning on general processor. The working frequency of general processor implementation is 3.46GHz with 64.0GB RAM. In addition, UCI benchmark data is selected for comparisons [62]. Write voltage V_w is set as $0.8V$ and read voltage V_r is set as $0.1V$ as well as duration time of $5ns$.

Performance Evaluation To evaluate the proposed RRAM network for machine learning, we use Iris in UCI dataset as benchmark. First we analyze the computation complexity in L2-norm-gradient based machine learning. Figure 6.3(a) shows the time consumption of CMOS-CPU based implementation. In simulation we need to optimize the value of 32 weights to minimize the L2-norm. 71.63% computation time is consumed by matrix-vector multiplication, and 26.41% time is consumed by L2-norm calculation. Therefore, the total computation time can be significantly reduced by accelerating these two operations.

To evaluate the RRAM network performance in Iris dataset, we use the proposed RRAM network to address matrix-vector multiplication and L2-norm calculation. For

Table 6.2: Time and energy comparison on UCI dataset.

Benchmark	Training sample	Hidden nodes	Classes	Specifications	CPU	CPU with RRAM
Iris	114	32	3	Time	0.0159s	0.0012s
				Multiplication	1.482J	1.39mJ
				L2-norm	0.546J	0.1μJ
				Total energy*	2.007J	0.042J
				Speed-up	-	13.47×
				Energy-saving	-	49.21×
Mushroom	7333	40	2	Time	0.0574s	0.005s
				Multiplication	3.315J	3.0mJ
				L2-norm	4.108J	7.584μJ
				Total energy*	7.462J	0.0357J
				Speed-up	-	11.48×
				Energy-saving	-	209.02×
Wine	160	40	3	Time	0.0409s	0.0029s
				Multiplication	3.562J	3.3mJ
				L2-norm	1.69J	0.3μJ
				Total energy*	5.317J	0.0639J
				Speed-up	-	13.96×
				Energy-saving	-	83.21×
Liver	317	232	2	Time	0.521s	0.0558s
				Multiplication	13.77J	12.9mJ
				L2-norm	0.546J	0.1μJ
				Total energy*	53.86J	0.12J
				Speed-up	-	9.34×
				Energy-saving	-	564.42×
Shuttle	14500	170	7	Time	3.35s	0.411s
				Multiplication	11.27J	10.56mJ
				L2-norm	0.546J	0.1μJ
				Total energy*	424.27J	0.0608J
				Speed-up	-	8.15×
				Energy-saving	-	7162.83×
Glass	192	105	7	Time	0.4s	0.0304s
				Multiplication	27.64J	25.90mJ
				L2-norm	24.22J	4.47μJ
				Total energy*	52J	0.109J
				Speed-up	-	13.16×
				Energy-saving	-	477.06×

* The total energy include multiplication, L2-norm and other operations (e.g. gradient calculation) performed in CPU.

the other operations (1.96% in Figure 6.3(a)), we still use CPU for calculation. As there are 32 weights to be updated in the benchmark, we design 32 RRAM-oscillator-

networks to compute E for each w simultaneously. In the experiment we need to repeat this procedure for 7 times to derive the minimum of L2-norm. The time consumption with RRAM network is shown in Figure 6.3(b). We can see that L2-norm becomes the dominant time-consuming operation. In addition, the matrix-vector multiplication and L2-norm take only $3\times$ as the other operations, which only take little time in machine learning.

Figure 6.3(c)(d) show the energy-efficiency achieved by the proposed RRAM network. The energy consumption of machine learning on RRAM network is significantly reduced. Note that each RRAM-oscillator-network requires $7.5\mu s$ to perform the one-time L2-norm calculation with $5.87\mu W$ according to Table 4.2. Note that matrix-vector multiplication and L2-norm calculation are dominant in CMOS-CPU based implementation. For Mushroom dataset, they take over 99% of energy consumption. When we

Figure 6.3: Time consumption proportion of Iris dataset (a) without RRAM network (b) with RRAM network; and energy consumption proportion of Mushroom dataset (c) without RRAM network (d) with RRAM network

map these operations on the RRAM network, the energy consumption of them is only 8.7%. We use 4 benchmarks in UCI dataset [62] to measure the speed-up and energy-saving, as shown in Table 6.2. Using the proposed RRAM network, we can achieve $10\times$ speed-up and more than $50\times$ energy-saving. Note that the improvement depends on the training size of benchmark. For example, the L2-norm calculation is the dominant operation in Mushroom and Liver so that we can save over 99.5% energy consumption when using the proposed RRAM network. For Iris and Wine, matrix-vector multiplication is dominant so that the energy-saving is below 99%.

6.1.2 Results of BCNN

6.1.2.1 BCNN Based Inference Acceleration on Passive Array

Experimental Settings

Baselines In the simulation, we have implemented different baselines for comparison using both MNIST and CIFAR-10 benchmarks. The detail of each baseline is listed below:

CPU: The BNN simulation is run in Matconvnet [98] on a computer server with 3.46GHz core and 64.0GB RAM. The BNN network is referred to Table 5.2.

Our design of Digital-RRAM: The RRAM device model for BNN is based on [37, 25, 26] with the resistance of RRAM set as 0.5M and 5M as on-state and off-state respectively with working frequency of 200MHz. Sense amplifier is based on the design of [99]. The BNN network is also referred to Table 5.2.

Others: GPU-based [66] Bitwise CNN implementations and FPGA-based [100], CMOS-ASIC based [101] and analog-RRAM based [44] conventional CNN implementations are selected for performance comparisons as well.

Network The overall network architecture of the bitwise CNN (called BNN) is shown in Table 5.2. It has six binary convolutional layers (BinConv), three max-pooling layers (MP) and three fully-connected layers (FC).

It takes a 32×32 RGB image as the input of the first layer. We use 128 sets of binary filters, and each set contains 3 binary filters to process the data from R, G, B channels, receptively. The width and height of each BinConv layer is fixed to 3×3 with stride of 1 and zero padding of 1. The BinConv layer performs the bitwise convolution between input feature maps and weights followed by the bitwise batch normalization and the binary activation. Note that the computation for FC layers can be treated as a convolution

with stride of 0 and zero padding of 0. Thus, we refer the matrix multiplication in FC layers to convolution as well.

As shown in Table 5.2, two cascaded convolutional layers form a convolutional block with equivalent 5×5 convolution window. This configuration provides more powerful representation capacity with less amount of weights compared with the direct 5×5 implementation. Meanwhile, binary batch-normalization is applied after the convolution to accelerate and stabilize the training. We adopt the binary activation to each BinConv layer and FC layer with the exception for the last FC layer. The output of the last FC layer is fed into the softmax layer [102] without binarization to generate a probabilistic distribution of 10 classes.

Besides CIFAR-10, we also show the experiment on MNIST dataset [103], with detailed parameters in Table 6.3, which is pre-trained on a server. Compare to parameters of CIFAR-10, we use conventional 5×5 convolutional cores in MNIST.

Performance Evaluation We first show accuracy comparison between the analogue RRAM and the digitized RRAM under device variation. We then show accuracy comparison between the conventional CNN with direct-truncated prevision and the proposed bitwise CNN. Various benchmarks such as CIFAR-10 [104] and MNIST [103] are used here.

Figure 6.4: RRAM configuration with voltage amplitude variation following Gaussian distribution.

Table 6.3: Bitwise CNN configuration of MNIST dataset.

Name	Filter/weight	Output Size
CONV-1	$5 \times 5 \times 1 \times 20$	$20 \times 28 \times 28$
MP-1	2×2	$20 \times 14 \times 14$
CONV-2	$5 \times 5 \times 20 \times 50$	$50 \times 14 \times 14$
MP-2	2×2	$50 \times 7 \times 7$
FC-1	2450×500	500
FC-2	500×10	10

Error under Device Variation In the previous discussion, the digitized RRAM-crossbar has better programming accuracy than the analogue RRAM-crossbar. Figure 6.4 shows a 200-time Monte-Carlo simulation of a single RRAM device programming process with different write voltage V_w , where the voltage amplitude is under Gaussian distribution on the $(3\sigma = 3\%V_w)$, and each column denotes a region of $5k\Omega$. It is clear that the digital RRAM-crossbar (only $500k\Omega$ and $5M\Omega$) can achieve a better uniformity than the analog RRAM-crossbar. The accuracy comparison against device variation on RRAM is shown in Figure 6.5(a). The percentage of device variation defines the maximum resistance offset, and each RRAM device follows standard deviation. Monte-Carlo is applied in the generation of device variation. In CIFAR-10, one can observe that when the device variation (RRAM resistance value) is more than 8%, there is a large output current error reported. For example, when the device variation reaches 29%, the digital RRAM can have an accuracy of 87.4% with only 4% decreased compared to no variations, better than the analog one with an accuracy of 84.8% with 7.6% decreased. In MNIST, the digital RRAM is always better even when the device variation is larger

Figure 6.5: (a) Accuracy comparison with analog RRAM under device variation (b) accuracy comparison with truncated CNN with approximation.

Figure 6.6: Scalability study of testing error rate under different energy consumption with (a) layer 2 output maps, and (b) layer 3 hidden nodes; (c) comparison with layer 2 output maps and layer 3 hidden nodes.

than 27%.

Error under Approximation For a precision-bit direct-truncated CNN, we use the conventional approach to train the full-precision (32-bit) CNN first, and then decrease the precision of all the weights in the network. The numerical experiment results of weights with different bit width is shown in Figure 6.5(b). Here the weights in the bitwise CNN is 1 bit, whose accuracy of is not changed. In CIFAR-10, although the

accuracy of the full precision (32-bit) can reach 92.4% in the direct-truncated CNN, the bit-width influences the accuracy a lot especially when the bit-width is smaller than 10. For example, when the precision decreases to 6-bit, the accuracy drops down to only about 11.8%. In MNIST, the accuracy of the direct-truncated CNN drops significantly when the bit-width is smaller than 6-bit. The results show that the proposed bitwise CNN can perform much better than the direct-truncated CNN.

Scalability Study To achieve a better energy-efficiency of BNN, we do the scalability study to find out the BNN parameters for both good testing accuracy and energy-efficiency. We use a 4-layer BNN on MNIST benchmark as a baseline (100% energy-efficiency, as shown in Table 6.3), and change the number of output maps of layer 2 (CONV-2, 50) and hidden nodes of layer 3 (FC-1, 500), as shown in Figure 6.6(a) and 6.6(b). For each energy-efficiency configuration, we do a 20-epoch training to make a fair comparison. When the number of hidden nodes or output maps decreases, the energy-efficiency is better but it will cause higher testing error rate. To summarize the scalability study, Figure 6.6(c) shows that the hidden nodes of layer 2 is more sensitive to testing accuracy. As a result, increasing the hidden nodes of layer 2 is better for higher accuracy while decreasing the hidden nodes of layer 3 is better for energy-efficiency.

Performance Comparison In this section, 1,000 images with 32×32 resolution in CIFAR-10 are selected to evaluate the performance among all implementations. Parameters including binary weights, and LUT for batch normalization have been configured by the training process. The detailed comparison is shown in Table 6.4 with numerical results including area, system throughput, computation time and energy consumption. Figure 6.7 shows an recognition example of CIFAR-10 dataset. In the numerical experiment, every batch comprises of 10 images and the results are calculated in parallel. The detailed result figure is included in supplementary file. The black square represents to the class that each image belongs.

The overall power comparison among all the implementations are shown in Table 6.4 under the similar accuracy of 91.4% in CIFAR-10. Compared to CPU-based and GPU-based bitwise CNN, the proposed digital-RRAM based implementation can achieve up to four-magnitude smaller power. Moreover, compared to FPGA-based and CMOS-ASIC based conventional CNN, the digital-RRAM based implementation is 4,155 times and 62 times smaller power.

In addition, we analyze the detailed power characteristics of the proposed digital-RRAM design in Figure 6.8. Firstly, we analyze the power distribution on convolution, batch normalization, pooling and activation, respectively. Figure 6.8(a) shows that

Figure 6.7: Image recognition example of CIFAR-10 dataset

89.05% of power is consumed by convolution, while 9.17% of power is for batch normalization, and the rest only takes 1.78%. Secondly, we analyze the power distribution on different bitwise CNN layers in Table 5.2. Results in Figure 6.8(b) show that CONV-6 consumes the most power 25.92%, while CONV-4 and CONV-2 also consume more than 23% of the total power. In general, there is over 98% of power consumed by CONV-2 to CONV-6 layers.

For the throughput performance, we use GOPS (Giga operations per second) to evaluate all the implementations. The proposed digital-RRAM can achieve 792 GOPS, which is 535 times and 1.61 times better than CPU-based and GPU-based implementations, respectively. It is also 12.78 times and 18.86 times better than FPGA-based and CMOS-ASIC. For energy-efficiency, the digital-RRAM achieves 176 TOPS/W, three-magnitude better than the CMOS-ASIC based CNN. In area-efficiency comparison, the digital-RRAM is 296 times better than CMOS-ASIC. The digital-RRAM is the best

Figure 6.8: Power consumption of bitwise CNN (a) for different operations (b) for different layers.

Figure 6.9: Performance with different number of CONV-4 output maps in CIFAR-10.

among all the implementations on both throughput and efficiency.

The design exploration of CIFAR-10 is shown in Figure 6.9. We change the number of CONV-4 output maps to find out the optimized parameter. The normalized energy consumption is referred to the configuration in Table 6.4. The result shows that the accuracy will not increase when the number is larger than 256. When it is lower than 256, the accuracy drops down even though it has better energy consumption and throughput. As a result, the parameters in Table 5.2 are optimal with specifications in Table 6.4.

6.1.2.2 BCNN Based Inference Acceleration on 1S1R Array

Experimental Settings

Baselines and Networks In the experiment, we have implemented different baselines for comparison using both CIFAR-10 and MNIST [103] benchmarks, the same as the benchmarks in Section 6.1.2.1. The detail of each baseline is listed below:

Bitwise CNN: We implement bitwise CNN on digital RRAM-crossbar with working frequency of 100MHz. Comparator on each BL is based on the design of [99]. In the 1S1R array, the drift-type RRAM in [37] is applied with $R_{LRS} = 500k\Omega$ and $R_{HRS} = 5M\Omega$. Meanwhile, the diffusive-type RRAM is taken from [35, 36] with $R_{on} = 9k\Omega$ and $R_{off} = 9k\Omega$. For performance comparison, CPU implementation is run in Matconvnet

Table 6.4: Performance comparison among different software and hardware implementations on CIFAR-10 BCNN.

Implementations	CPU (Mat-convnet)	GPU [66]	FPGA [100]	CMOS-ASIC [101]	Analog RRAM [44]	Digital passive RRAM
Network category	Bitwise CNN	Bitwise CNN	CNN	CNN	CNN	Bitwise CNN
Frequency	3.46GHz	1.05GHz	100MHz	250MHz	-	100MHz
Power	130W	170W	18.7W	278mW	-	4.5mW
Area (mm ²)	240	601	-	12.25	-	0.78
System throughput	1.48 GOPS	493 GOPS	62 GOPS	42 GOPS	-	792 GOPS
Frame per second	1.2FPS	400FPS	46FPS	35FPS	-	643FPS
Energy-efficiency (GOPS/W)	0.011	2.9	3.32	151	2000	176000
Area-efficiency (GOPS/mm ²)	0.006	0.82	-	3.43	-	1015

- refers to the data not reported.

on a computer server with 3.46GHz core and 64.0GB RAM. The GPU based design is referred to [66], and a CMOS-ASIC design in 40nm technology is also realized.

Figure 6.10: Sneak-path-free effect in 1S1R implementation for write operation.

Figure 6.11: Sneak-path-free effect in 1S1R implementation for read operation.

Conventional CNN: FPGA based [100] and CMOS-ASIC based [105] implementations are compared. Moreover, an analog RRAM-crossbar based realization are simulated.

RRAM Device Simulation To illustrate the sneak-path-free effect of the 1S1R implementation, we use SPICE simulation with the same input voltages for both Figure 2.14(a) and 2.14(c). In the write operation, we set the target $R_{i,j}$ as HRS and the sneak-path RRAMs $R_{i,j-1}$, $R_{i+1,j-1}$, $R_{i+1,j}$ are HRS, LRS and LRS, respectively. By applying $V_{WL,i} = 1$ and $V_{BL,j} = 0$ with other nets floating, the sneak-path RRAM $R_{i,j-1}$ in direct connection is changed from $5M\Omega$ to $1.4M\Omega$, while the one in the 1S1R implementation remains the original resistance shown in Figure 6.10. In the read operation, we set $R_{i,j}$ as HRS and the sneak-path RRAMs are all LRS. According to the results in Figure 6.11, the direct connection implementation will read $3R_{LRS}/R_{HRS}$ due to the sneak-path effect. In contrast, though it takes $500ns$ to turn on the selector with read voltage, the target RRAM can be precisely read out.

Although the $500ns$ read time of the proposed 1S1R is much longer than $10ns$ of direct connection, the total read energy will not increase much. As shown in Figure 6.11, because the selector in LRS branch can be turned on faster than the one in RHS branch, the RHS/LRS ratio in the read operation can reach 44 instead of the ideal value 10 ($5M/500k$). As a result, using selector can achieve a larger differences between reading LRS and reading RHS.

In addition, we show the RRAM resistance variation with different write voltage

Figure 6.12: Programing inaccuracy for different target resistances with voltage amplitude variation following Gaussian distribution

V_w in Figure 6.12. We use 5 sets of ideal V_w from 0.6V to 1.0V following Gaussian distribution ($3\sigma = 5\%V_w$). 2000 data points are simulated to show the distribution of configured RRAM resistance value in each set. Each column in Figure 6.12 represents a resistance range of $10k\Omega$. The results show that the digital RRAM (LRS with 1.0V here) has a much better uniformity with input variation compared to analog RRAM like [48], leading a better programming accuracy.

Performance Comparison We select CIFAR-10 as the benchmark to evaluate all the implementations. Table 6.5 summarize the performance under similar classification accuracy of 91.4%.

Power Compared to CPU-based and GPU-based bitwise CNN, the proposed digital-RRAM based implementation can achieve up to four-magnitude smaller power. Compared to simulation result of CMOS-ASIC bitwise CNN, it has 3.17 times smaller power. Moreover, compared to FPGA-based and CMOS-ASIC based conventional CNN, the digital-RRAM based implementation is 2968 times and 44.1 times smaller power, respectively.

In addition, we analyze the detailed power characteristics of the proposed digital-RRAM design in Figure 6.8. Firstly, we analyze the power distribution on convolu-

Table 6.5: Performance comparison under different software and hardware implementations

Implementations	CPU (Mat-convnet)	GPU [66]	FPGA [100]	CMOS-ASIC [101]	CMOS-ASIC simulation	Digital 1S1R RRAM
Network category	Bitwise CNN	Bitwise CNN	CNN	CNN	Bitwise CNN	Bitwise CNN
Frequency	3.46GHz	1.05GHz	100MHz	250MHz	30MHz	100MHz
Power	130W	170W	18.7W	278mW	20mW	6.3mW
Area (mm ²)	240	601	-	12.25	10	1.02
System throughput	1.48 GOPS	493 GOPS	62 GOPS	42 GOPS	621 GOPS	792 GOPS
Energy-efficiency (GOPS/W)	0.011	2.9	3.32	151	31000	126000
Area-efficiency (GOPS/mm ²)	0.006	0.82	-	3.43	62.1	776

- refers to the data not reported.

tion, batch normalization, pooling and activation, respectively. Figure 6.8(a) shows that 89.05% of power is consumed by convolution, while 9.17% of power is for batch normalization, and the rest only takes 1.78%. Secondly, we analyze the power distribution on different bitwise CNN layers in Table 5.2. Results in Figure 6.8(b) show that CONV-6 consumes the most power 25.92%, while CONV-4 and CONV-2 also consume more than 23% of the total power. In general, there is over 98% of power consumed by CONV-2 to CONV-6 layers.

Throughput and Energy-efficiency For the throughput performance, we use giga operations per second (GOPS) to evaluate all the implementations. The proposed digital-RRAM can achieve 792 GOPS, which is 535 times, 1.61 times and 1.28 times better than CPU-based, GPU-based and CMOS-ASIC-based bitwise CNN, respectively. It is also 12.78 times and 18.86 times better than FPGA-based and CMOS-ASIC-based CNN. For energy-efficiency, the digital-RRAM achieves 126 TOPS/W, three-magnitude better than the CMOS-ASIC based CNN. In area-efficiency comparison, the digital-RRAM is 296 times better than CMOS-ASIC based CNN as well. The digital-RRAM is the best among all the implementations on both throughput and efficiency.

6.2 Results on 3D CMOS-RRAM

6.2.1 Results of SLFN

6.2.1.1 On-Chip Design for SLFN Based Face Recognition

Experimental Settings In the experiment, we have implemented different baselines for performance comparisons. The detail of each baseline is listed below:

Baseline 1: General processor. The general process implementation is based on Matlab on a computer server with 3.46GHz core and 64.0GB RAM.

Baseline 2: Graphics Processing Unit (GPU). The GPU implementation is performed by Matlab GPU parallel toolbox on the same server. An Nvidia GeForce GTX 970 is used for the acceleration of matrix-vector multiplication operations for learning on neuron network.

Figure 6.13: Scalability study of hardware performance with different hidden node numbers for: (a) area; (b) delay; (c) energy and (d) energy-delay-product (EDP)

Baseline 3: 3D-CMOS-ASIC. The 3D-CMOS-ASIC implementation with proposed architecture is done by Verilog with 1GHz working frequency based on CMOS 65nm low power PDK. Power, area and frequency are evaluated through Synopsys DC compiler (D-2010.03-SP2). Through-silicon via (TSV) model in [106] is included for area and power evaluation. 512 vertical TSVs are assumed between layers to support communication and parallel computations [107].

Proposed 3D-RRAM-CMOS: The settings of CMOS evaluation and TSV model are the same as baseline 3. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as $1k\Omega$ and $1M\Omega$ as on-state and off-state resistance respectively according to [81] with working frequency of 200MHz.

Performance Evaluation

Scalability Analysis To evaluate the proposed 3D multi-layer CMOS-RRAM architecture, we performed the scalability analysis of energy, delay and area on glass UCI dataset [62]. In SLFN, the number of hidden nodes may change depending on the accu-

Table 6.6: Testing accuracy of machine learning techniques under different dataset and configurations (normalized to all 32 bits)

Datasets	Size	Features	Classes	Bit width	Accuracy (%)	
					SVM $\ddagger$	SLFN
Glass	214	9	6	4	100.07	100.00
				5	93.88	99.30
				6	99.82	99.30
Iris	150	4	3	4	98.44	94.12
				5	100.00	94.18
				6	100.00	100.00
Seeds	210	7	3	4	97.98	82.59
				5	99.00	91.05
				6	99.00	98.51
Arrhythmia	179	13	3	4	96.77	97.67
				5	99.18	98.83
				6	99.24	100.00
Letter	20,000	16	7	4	97.26	53.28
				5	98.29	89.73
				6	99.55	96.13
CIFAR-10	60,000	1,600 $\dagger$	10	4	98.75	95.71
				5	99.31	97.06
				6	99.31	99.33

$\dagger$ 1600 features extracted from 60,000 32×32 color images with 10 classes.

$\ddagger$ Least-square SVM is used for comparison.

Figure 6.14: Energy-saving comparison under different bit-width of CMOS and RRAM with the same accuracy requirement.

racy requirement. As a result, the improvement of proposed accelerator with different L from 64 to 256 is evaluated as shown in Figure 6.13. With the increasing L , more computing units are designed in 3D-CMOS-ASIC and RRAM-crossbar to evaluate the performance. When L reaches 256, RRAM-crossbar can achieve $2.1\times$ area-saving and $10.02\times$ energy-saving compared to 3D-CMOS-ASIC. In Figure 6.13(d), energy-delay-product (EDP) of RRAM-crossbar increases faster than 3D-CMOS-ASIC. As the size of RRAM-crossbar is proportional to square of L , the EDP improvement of RRAM-crossbar is less with larger L . However, it still shows great advantage in EDP with $51\times$ better than 3D-CMOS-ASIC when $L = 500$, which is large number of hidden node for glass benchmark of 9 features.

Bit-width Configuration Analysis Table 6.6 shows the testing accuracy under different datasets [62, 108], and configurations for machine learning of support vector machine (SVM) and single layer feed-forward neuron network (SLFN). It shows that accuracy of classification is not very sensitive to the RRAM configuration bits. For example, the accuracy of Iris dataset is working with negligible accuracy at 5 RRAM bit-width. When the RRAM bit-width increased to 6, it performs the same as 32 bit-width configurations. Similar observation is found in [95] by truncating algorithms with limited

Table 6.7: Performance comparison under different software and hardware implementations.

Implementation	Hardware specifications	Computation	Operations	Time	Energy
CPU (Matlab)	3.46GHz 240mm ² 130W	Feature extraction	Sort	1473.2s	191.52kJ
			Multiplication	736.6s	95.76kJ
		Input layer	Multiplication	729.79s	94.87kJ
		L2-norm	Division	294.34s	38.26kJ
			Multiplication	1667.95s	216.83kJ
		Output layer	Multiplication	750.3s	97.54kJ
		Overall	-	5651.88s	734.78kJ
3D-CMOS-ASIC [†]	1.0GHz 1.86mm ² 39.41W	Feature extraction	Sort	216.65s	78J
			Multiplication	97.43s	3840J
		Input layer	Multiplication	96.53s	3800J
		L2-norm	Division	43.29s	15J
			Multiplication	220.62s	8690J
		Output layer	Multiplication	99.25s	3910J
		Overall	-	773.77s	20.34kJ
3D-CMOS-RRAM [†]	1.0GHz 1.86mm ² 39.41W	Feature extraction	Sort	216.65s	78J
			Multiplication	22.43s	293J
		Input layer	Multiplication	22.22s	291J
		L2-norm	Division	43.29s	15J
			Multiplication	50.79s	670J
		Output layer	Multiplication	22.85s	300J
		Overall	-	378.22s	1643J
		Improvement	-	14.94×	447.17×

[†]6 bit-width configuration is implemented for both CMOS and RRAM.

precision for better energy efficiency. Please note that training data and weight related parameters are quantized to perform matrix-vector multiplication on RRAM crossbar accelerator. Figure 6.14 shows the energy comparisons under different bit-width configurations for CMOS and RRAM under the same accuracy requirements. An average of 4.5× energy saving can be achieved for the same number of bit-width configurations. The energy consumption is normalized by the CMOS 4 bit-width configuration. Furthermore, we can observe that not always smaller number of bits achieves better energy saving. Fewer number of bit-width may require much larger neuron network to perform required classification accuracy. As a result, its energy consumption increases.

Performance Analysis Figure 6.15 shows the classification values on image data [108] with an example of 5 classes. As mentioned before, the index with maximum values (highlighted in red) is selected to indicate the class of test case. A few sample

Test cases/ Classes	Test case 1	Test case 2	Test case 3		Class 1 ship
Class 1	-3.3363	-0.0037	2.2211		Class 2 dog
Class 2	-3.1661	0.6737	1.2081		Class 3 airplane
Class 3	-3.5008	0.0613	1.4670		Class 4 bird
Class 4	-3.4521	-0.0527	1.5498		Class 5 cat
Class 5	-3.0085	-0.1861	1.0764		
•	•	•	•		
•	•	•	•		
•	•	•	•		

Figure 6.15: On-line machine-learning for image recognition on the proposed 3D multi-layer CMOS-RRAM accelerator using benchmark CIFAR-10

images are selected. Please note that 50,000 and 10,000 images are used for training and testing with 10 classes. In Table 6.7, performance comparisons among Matlab, 3D-CMOS-ASIC and 3D multi-layer CMOS-RRAM accelerator are presented, and the acceleration of each procedure based on the formula described is also addressed. Among the three implementations, 3D multi-layer CMOS-RRAM accelerator performs the best in area, energy and speed. Compared to Matlab implementation, it achieves $14.94\times$ speed-up, $447.17\times$ energy-saving and $164.38\times$ area-saving. We also design a 3D-CMOS-ASIC implementation with similar structure as 3D multi-layer CMOS-RRAM accelerator with better performance compared to Matlab. The proposed 3D multi-layer CMOS-RRAM 3D accelerator is $2.05\times$ speed-up, $12.38\times$ energy-saving and $1.28\times$ area-saving compared to 3D-CMOS-ASIC. To compare the performance with GPU, we also implemented the same code using Matlab GPU parallel toolbox. It takes 1163.42s for training benchmark CIFAR-10, which is $4.858\times$ faster than CPU. Comparing to our proposed 3D multi-layer CMOS-RRAM architecture, our work is $3.07\times$ speed-up and $162.86\times$ energy saving. Detailed comparisons of each step is not shown due to the limited space of table.

6.2.2 Results of TNN

6.2.2.1 TNN Based On-Chip Design on 3D Multi-Layer Architecture

Experimental Settings In the experiment, we have implemented different baselines for performance comparisons. The detail of each baseline is listed below:

Baseline 1: General CPU processor. The general process implementation is based on Matlab with optimized C-program. The computer server is with 6 cores of 3.46GHz

and 64.0GB RAM.

Baseline 2: General GPU processor. The general-purpose GPU implementation is based on the optimized C-program and Matlab parallel computing toolbox with CUDA-enabled Quadro 5000 GPU [109].

Baseline 3: 3D CMOS-ASIC. The 3D CMOS-ASIC implementation with proposed architecture is done by Verilog with 1GHz working frequency based on CMOS 65nm low power PDK. Power, area and frequency are evaluated through Synopsys DC compiler (D-2010.03-SP2). Through-silicon via (TSV) area, power and delay are evaluated based on Simulator DESTINY [88] and fine-grained TSV model CACTI-3DD [110]. The buffer size of the top layer is set 128MB to store tensor cores with 256 bits data width. The TSV area is estimated to be $25.0 \mu m^2$ with capacitance of $21 fF$.

Proposed 3D CMOS-RRAM: The settings of CMOS evaluation and TSV model are the same as baseline 2. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as 500K and 5M as on-state and off-state resistance and 2V SET/RESET voltage according to [37] with working frequency of 200MHz. The CMOS and RRAM integration is evaluated based on [26].

To evaluate the proposed architecture, we apply UCI [62] and MNIST [103] dataset to analyze the accelerator scalability, model configuration analysis and performance analysis. The model configuration is performed on Matlab first using Tensor-train toolbox [111] before mapping on the 3D CMOS-RRAM architecture. To evaluate the model compression, we compare our method with SVD based node pruned method [112] and general neural network [14]. The energy consumption and speed-up are also evaluated. Note that the code for performance comparisons is based on optimized C-Program and deployed as the mex-file in the Matlab environment.

Performance Evaluation

3D Multi-layer CMOS-RRAM Accelerator Scalability Analysis Since neural network process requires frequent network weights reading, memory read latency optimization configuration is set to generate RRAM memory architecture. By adopting 3D implementation, Simulation results show that memory read and write bandwidth can be significantly improved by 51.53% and 6.51% respectively comparing to 2D implementation. For smaller number of hidden nodes, read/write bandwidth is still improved but the bottleneck shifts to the latency of memory logic control.

To evaluate the proposed 3D multi-layer CMOS-RRAM architecture, we perform the scalability analysis of energy, delay and area on MNIST dataset [103]. This dataset is applied to multi-layer neural network and the number of hidden nodes may change

Table 6.8: Bandwidth improvement under different number of hidden nodes for MNIST dataset.

Hidden node [†] (L)	256	512	1024	2048	4096
Memory required (MB)	1.025	2.55	7.10	22.20	76.41
Memory set (MB)	2M	4M	8M	32M	128M
Write Bandwidth Imp.	1.14%	0.35%	0.60%	3.12%	6.51%
Read Bandwidth Imp.	5.02%	6.07%	9.34%	20.65%	51.53%

[†]4-layer neural network with 3 full-connected layer $784 \times L$, $L \times L$ and $L \times 10$.

depending on the accuracy requirement. As a result, the improvement of proposed accelerator with different L from 32 to 2048 is evaluated as shown in Figure 6.16. With the increasing L , more computing units are designed in 3D CMOS-ASIC and RRAM-crossbar to evaluate the performance. The neural network is defined as a 4-layer network

Figure 6.16: Scalability study of hardware performance with different hidden node numbers for: (a) area; (b) delay; (c) energy and (d) energy-delay-product

Figure 6.17: Testing time and accuracy comparison between tensorized neural network (TTN) and general neural network (Gen.) with varying number of hidden nodes.

with weights $784 \times L$, $L \times L$ and $L \times 10$. For computation delay, GPU, 3D CMOS-ASIC and 3D CMOS-RRAM are close when $L = 2048$ according to Figure 6.16(b). When L reaches 256, 3D CMOS-RRAM can achieve $7.56\times$ area-saving and $3.21\times$ energy-saving compared to 3D CMOS-ASIC. Although the computational complexity is not linearly related to the number of hidden node numbers, both energy consumption and energy-delay-product (EDP) of RRAM-crossbar increase with the rising number of hidden node. According to Figure 6.16(d), the advantage of the hybrid accelerator becomes smaller when the hidden node increases, but it can still have a $5.49\times$ better EDP compared to the 3D CMOS-ASIC when the hidden node number is 2048.

Model Configuration Analysis Figure 6.17 shows the testing accuracy and running time comparisons for MNIST dataset. It shows a clear trend of accuracy improvement with increasing number of hidden nodes. The running time between TTN and

Table 6.9: Model Compression under different number of hidden nodes and tensor ranks on MNIST dataset.

Number of hidden nodes [†]	32	64	128	256	512	1024	2048
Compression	5.50	4.76	3.74	3.23	3.01	4.00	8.18
Rank [‡]	15	20	25	30	35	40	45
Compression	25.19	22.51	20.34	17.59	14.85	12.86	8.63
Accuracy (%)	90.42	90.56	91.41	91.67	93.47	93.32	93.86

[‡] Number of hidden nodes are all fixed to 2048 with 4 fully connected layers.

[†] All tensor Rank is initialized to 50.

Table 6.10: Testing accuracy of learning techniques under different dataset and bit-width configuration.

Datasets	32-bit Acc. (%) & Compr.		4 bit Acc. (%) & Compr.		5 bit Acc. (%) & Compr.		6 bit Acc. (%) & Compr.	
Glass	88.6	1.32	89.22	10.56	83.18	8.45	88.44	7.04
Iris	96.8	1.12	95.29	8.96	96.8	7.17	96.8	5.97
Diabets	71	3.14	69.55	25.12	69.4	20.10	71.00	16.75
Adult	78.8	1.87	75.46	14.96	78.15	11.97	78.20	9.97
Leuke	88.9	1.82	85.57	14.56	87.38	11.65	88.50	9.71
MNIST	94.38	4.18	91.28	33.44	92.79	26.75	94.08	22.29

general NN is almost the same. This is due to the relative large rank $r = 50$ and computation cost of $O(dr^2n \max(N, L))$. Such tensor-train based neural network achieve $4\times$ and $8.18\times$ model compression within 2% accuracy loss under 1024 and 2048 number of hidden nodes respectively. Details on model compression are shown in Table 6.9. From Table 6.9, we can observe that the compression rate is directly connected with the rank r , where the memory storage can be simplified as dnr^2 from $\sum_{k=1}^d n_k r_{k-1} r_k$ but not directly link to the number of hidden nodes. We also observe that by setting tensor core rank to 35, $14.85\times$ model compression can be achieved with acceptable accuracy loss. Therefore, initialization of a low rank core and the SVD split of supercore in MALS algorithm are important steps to reduce the core rank and increase compression rate.

Bit-width Configuration Analysis To implement the whole neural network on the proposed 3D multi-layer CMOS-RRAM accelerator, the precision of real values requires a careful evaluation. Compared to the software double precision floating point format (64-bit), values are truncated into finite precision. By using the greedy search method, an optimal point for hardware resource (small bit-width) and testing accuracy can be achieved.

Our tensor-train based neural network compression techniques can work with low-precision value techniques to further reduce the data storage. Table 6.10 shows the testing accuracy by adopting different bit-width on UCI datasets [62] and MNIST [103]. It shows that accuracy of classification is not very sensitive to the RRAM configuration bits for UCI dataset. For example, the accuracy of Iris dataset is working well with negligible accuracy at 5 RRAM bit-width. When the RRAM bit-width increased to 6, it performs the same as 32 bit-width configurations. Please note that the best configuration of quantized model weights varies for different datasets and requires careful evaluation.

Performance Analysis In Table 6.11, performance comparisons among C-Program Optimized CPU performance, GPU performance, 3D CMOS-ASIC and 3D multi-layer CMOS-RRAM accelerator are presented for 10,000 testing images. The acceleration of each layer is also presented for 3 layers (784×2048 , 2048×2048 and 2048×10). Please note that the dimension of weight matrices are decomposed into $[4 \ 4 \ 7 \ 7]$ and $[4 \ 4 \ 8 \ 8]$ with 6 bit-width and maximum rank 6. The compression rate is $22.29\times$ and $4.18\times$ with and without bit-truncation. Here the compression rate is the ratio of volume of weights before and after compression. Among the four implementations, 3D multi-layer CMOS-RRAM accelerator performs the best in area, energy and speed. Compared to CPU, it achieves $6.37\times$ speed-up, $2612\times$ energy-saving and $233.92\times$ area-saving. For GPU based implementation, our proposed 3D CMOS-RRAM architecture achieves $1.43\times$ speed-up and $694.68\times$ energy-saving. We also design a 3D CMOS-ASIC implementation with similar structure as 3D multi-layer CMOS-RRAM accelerator with better performance compared to CPU and GPU based implementations. The proposed 3D multi-layer CMOS-RRAM 3D accelerator is $1.283\times$ speed-up, $4.276\times$ energy-saving and $9.339\times$ area-saving compared to 3D CMOS-ASIC.

The throughput and energy efficiency for these four cases are also summarized in Table III. For energy efficiency, our proposed accelerator can achieve 1499.83GOPS/W,

Table 6.11: Performance comparison under different hardware implementations on MNIST dataset with 10,000 testing images.

Implementation	Hardware specifications	Computations	Time	Speed-up	Energy	Energy-saving
CPU [113]	3.46GHz 240mm ² 130W 74.64GOPS	Layer 1	0.44s	-	57.23J	-
		Layer 2	0.97s	-	125.74J	-
		Layer 3	0.045s	-	5.82J	-
		Overall	1.45s	-	188.8J	-
GPU [109]	513MHz 529mm ² 152W 328.41GOPS	Layer 1	0.04s	11.06×	4.05J	9.46×
		Layer 2	0.289s	3.36×	43.78J	2.87×
		Layer 3	2.4ms	18.67×	0.3648J	15.96×
		Overall	0.33s	4.40×	50.19J	3.76×
3D CMOS-ASIC Architecture	1.0GHz 9.582mm ² 1.037W 367.43GOPS	Layer 1	0.032s	13.9×	33.3mJ	1711.8×
		Layer 2	0.26s	3.71×	0.276J	455.4×
		Layer 3	2.4ms	18.4×	2.6mJ	2262×
		Overall	0.295s	4.92×	0.312J	604.9×
3D CMOS-RRAM Architecture	100MHz 1.026mm ² 0.317W 475.45GOPS	Layer 1	0.025s	17.77×	7.9mJ	7286×
		Layer 2	0.20s	4.80×	0.64J	25.86×
		Layer 3	1.7ms	25.86×	0.5mJ	11000×
		Overall	0.23s	6.37×	0.072J	2612×

†4-layer neural network with weights 784×2048 , 2048×2048 and 2048×10 .

which has $4.30\times$ better energy efficiency comparing to 3D the CMOS-ASIC result (347.29GOPS/W). In comparison to our GPU baseline, it has $694.37\times$ better energy efficiency comparing to NVIDIA Quadro 5000. For a newer GPU device (NVIDIA Tesla K40), which can achieve 1092GFLOPS and consume 235W [109], our proposed accelerator has $347.49\times$ energy efficiency improvement.

6.2.2.2 TNN Based Distributed On-Chip Design on 3D Single-Layer Architecture

Experiment Settings In the experiment, we have implemented different baselines for performance comparisons. The detail of each baseline is listed below:

Baseline 1: General processor (CPU/GPU). The general processor implementation is based on optimized C-program on a computer server with 6 cores of $3.46GHz$ and $64.0GB$ RAM. A Nvidia GeForce GTX 970 is used as reference for the GPU performance comparison.

Baseline 2: 2D CMOS-ASIC. The 2D CMOS-ASIC implementation with proposed architecture is done by Verilog with $1GHz$ working frequency based on CMOS 65nm low power PDK. Power, area and frequency are evaluated through Synopsys DC compiler (D-2010.03-SP2).

Baseline 3: 3D CMOS-ASIC. The 3D CMOS-ASIC implementation is the same as 2D version but features with TSV communication. Through-silicon via (TSV) area, power and delay are evaluated based on Simulator DESTINY [88] and fine-grained TSV model CACTI-3DD [110]. The buffer size of the top layer is SRAM based and set $128MB$ to store tensor cores. The TSV area is estimated to be $25.0\mu m^2$ with capacitance of $21fF$. Through-silicon via (TSV) model in [106] is included for area and power evaluation. 512 vertical TSVs are assumed between layers to support communication and parallel computations [107].

Proposed 3D CMOS-RRAM: The settings of CMOS evaluation and TSV model are the same as baseline 3. For the RRAM-crossbar design evaluation, the resistance of RRAM is set as $1K\Omega$ and $1M\Omega$ as on-state and off-state resistance and 2V SET/RESET voltage according to [81] with working frequency of $200MHz$. The CMOS and RRAM integration is evaluated based on [26].

To evaluate the proposed architecture, we apply CIFAR-10 [104] and MNIST [103] dataset to analyze the neural network compression and architecture performance analysis. The energy-efficient neural network is designed on Matlab first using Tensor-train toolbox [111] and tensor network [74]. After determining the network configuration and compression rate, it is then fully optimized using C-program as CPU based result. Then we apply non-uniform quantization to perform tensor core quantization before mapping

Table 6.12: Model Compression under different number of hidden nodes and tensor ranks on MNIST dataset.

Rank[‡]	2	4	6	8	10	12
Compression ($\times$)	69.73	61.70	52.53	43.54	35.71	29.30
Q-Compression ($\times$)	223.4	179.49	152.81	139.33	114.23	93.76
Error Rate (%)	4.28	4.14	4.03	4.08	3.97	3.79

[‡] Number of hidden nodes are all fixed to 1024 with 1 hidden layers.

on the 3D CMOS-RRAM architecture. To evaluate the model compression, we compare our method with SVD based node pruned method [112] and general neural network [14]. The energy efficiency and speed-up are also evaluated.

Energy-efficient Tensor-compressed Neural Network To achieve high compression rate, we firstly apply non-uniform quantization for tensor core weights. As shown in Figure 6.18, the probability density function of tensor core weights can be modeled as Gaussian distribution. For such known pdf, we can effectively find the optimal level representative values with minimized mean square error. Figure 6.19 shows the trade-off between accuracy, bit-width, compression rate on MNIST dataset with 3 layer neural network. Table 6.12 shows the clear effect of quantization on the compression rate. In generally, bit-width quantization can help improve $3\times$ more compression rate on neural network.

To have a fair comparison with [114], we also adopted LeNet-300-100 network [103], which is a two hidden layer fully-connected layers (784×300 , 300×100 , 100×10).

Figure 6.18: Tensorized neural network layer 1 weight and layer 2 weight histogram with approximate Gaussian distribution.

Figure 6.19: Compression rate and accuracy with increasing bit-width.

Under such configuration, [114] can achieve $40\times$ compression with error rate 1.58 % using quantization (6-bit precision), pruning and Huffman coding techniques. By adopting 9 bit-width configuration on tensorized layer, we can achieve even more compression rate ($142\times$) with maintained accuracy (1.63%).

We also apply the proposed TNN to CIFAR-10 dataset. CIFAR-10 dataset consists of 60,000 images with size of $32\times32\times3$ under 10 different classes. We adopt LeNet-like neural network [103] and we use a two layer fully connected TNN to replace the last two fully-connected layers, which is 512×64 and 64×10 . By adopting non-uniform tensor-core bit quantization (8 bit-width), we can achieve $17.6\times$ compression on fully connected layers and $2\times$ on the whole network respectively with the same accuracy loss (2%).

Table 6.13: Bandwidth improvement under different number of hidden nodes for MNIST dataset.

Hidden node $\dagger(L)$	256	512	1024	2048	4096
Memory required (MB)	1.025	2.55	7.10	22.20	76.41
Memory set (MB)	2M	4	8	32	128
Write bandwidth improvement	1.14%	0.35%	0.60%	3.12%	6.51%
Read bandwidth improvement	5.02%	6.07%	9.34%	20.65%	51.53%

$\dagger$ 4-layer neural network with 3 full-connected layer $784\times L$, $L\times L$ and $L\times10$.

Table 6.14: Performance comparison under different hardware implementations on MNIST dataset (10K test images).

Implementations	CPU	GPU	2D CMOS-ASIC	3D CMOS-ASIC	3D CMOS-RRAM	
Frequency	3.46GHz	200MHz	1.0GHz	1.0GHz	100MHz	
Power	130W	152W	779mW	649mW	351mW	
Area (mm ²)	240	529	12.69	4.63	0.61	
System throughput	73.63 GOPS	256.32 GOPS	244.53 GOPS	286.44 GOPS	370.64 GOPS	
Energy-efficiency	0.57 GOPS/W	1.69 GOPS/W	313.98 GOPS/W	441.36 GOPS/W	1055.95W GOPS/W	
Network compressed?	Yes	Yes	Yes	Yes	No	Yes
Time	1.452s	0.213s	0.223s	0.191s	0.747s	0.1473s
Energy	188.79J	32.376J	0.1738J	0.1237J	0.2621J	0.0517J

†10k test images on 3-layer neural network with two full-connected layer 784×1024 and 1024×10 . 6 bit-width is adopted for tensor core weights with maximum rank 6.

Energy-efficient 3D CMOS-RRAM Architecture Since neural network process requires frequent network weights reading, memory read latency optimization configuration is set to generate RRAM memory architecture. By adopting 3D implementation, Simulation results show that memory read and write bandwidth can be improved by 51.53% and 6.51% respectively comparing to 2D implementation as shown in Table 6.13. For smaller number of hidden nodes, read/write bandwidth is still improved but the bottleneck shifts to the latency of memory logic control.

In Table 6.14, performance comparisons among general processors, 2D CMOS-ASIC, 3D CMOS-ASIC and 3D CMOS-RRAM accelerator are presented, and the acceleration of each layer is also presented for 2 layers (784×1024 and 1024×10). Please note that input weight is decomposed into $[4 \ 4 \ 7 \ 7]$ and $[4 \ 4 \ 8 \ 8]$ with maximum rank 6 and 6 bit-width. Among the four implementations, 3D CMOS-RRAM accelerator performs the best in area, energy and speed. Compared to 2D CMOS implementation, it achieves $1.515 \times$ speed-up, $3.362 \times$ energy-saving and $20.80 \times$ area-saving. We also design a 3D CMOS-ASIC implementation with similar structure as 3D CMOS-RRAM accelerator with better performance compared to 2D CMOS-ASIC. The proposed 3D CMOS-RRAM accelerator is $1.294 \times$ speed-up, $2.393 \times$ energy-saving and $7.59 \times$ area-saving compared to 3D CMOS-ASIC. We also performed the mapping of uncompressed neural network on 3D CMOS-RRAM architecture. In addition, for CIFAR-10 dataset, we estimate $2.79 \times$ and $2.23 \times$ energy-saving comparing to 2D CMOS-ASIC and 3D

CMOS-ASIC architecture respectively.

For energy efficiency, our proposed accelerator can achieve 1055.95GOPS/W, which is equivalent to 7.661TOPS/W for uncompressed neural network. Our proposed TNN accelerator can also achieve $2.39\times$ better energy efficiency comparing to 3D CMOS-ASIC result (441.36GOPS/W); and $227.24\times$ better energy efficiency comparing to Nvidia Tesla K40, which can achieve 1092GFLOPS and consume 235W.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

To conclude, this thesis has shown a thorough study on RRAM based non-volatile in-memory architecture towards machine learning applications from circuit level, to architecture level, and all the way to system level.

For the circuit level, on one hand we use NVM-SPICE to simulate matrix-vector multiplication acceleration by binary RRAM-crossbar. By programming all the elements of a matrix into RRAM resistance in the crossbar array, we can use crossbar wordline voltages to represent another matrix, and the multiplication result can be denoted by the merging current on bitlines. In this mapping scheme, both RRAM resistance and crossbar I/O are in binary format so that the computation accuracy can be guaranteed. Simulation results show that the proposed architecture has shown $2.86\times$ faster speed, $154\times$ better energy efficiency, and $100\times$ smaller area when compared to the same design by CMOS-based ASIC. On the other hand, we use a Verilog-A model to build an RRAM-based voltage-controlled oscillator circuit in Cadence. In a coupled network by RRAM-based oscillator, the simulation results can be fitted as L2-norm calculation. Compared to traditional CMOS based oscillator circuit, it has a much simpler structure with smaller area and better energy-efficiency.

For the architecture level, we have developed a distributed in-memory architecture (XIMA) and 3D CMOS-RRAM architecture. For XIMA, we use both RRAM-crossbar for data storage and computing engine. We use a data block and logic block to form a data-logic pair, and all the pairs are located distributively. As a result, the communication bandwidth can be significantly improved because the computing engine only accesses the data from pairs. To achieve the in-memory computing, we design a control bus in each data-logic pair for communication among processor, data and logic block.

In addition, communication protocol between processor and control bus is re-defined for the in-memory computing implementation. For 3D CMOS-RRAM architecture, we developed two schemes: single-layer and multi-layer. The single-layer architecture uses RRAM as vias connecting the top-layer wordlines and bottom-layer bitlines. All the other CMOS logics are implemented in the bottom-layer as well. The multi-layer architecture uses TSVs connecting the first-layer RRAM-based data buffer, the second-layer RRAM-based logic computing and the third-layer CMOS. Such a 3D architecture can significantly improve the throughput as well as the area-efficiency for the hybrid RRAM/CMOS computing system.

For the system level, for the distributed in-memory architecture (XIMA), we have accelerated three machine learning algorithms. The learning and inference procedures of single-layer feed-forward neural network (SLFN) have been optimized and partially mapped on the passive binary RRAM-crossbar. In addition, we mapped the binary convolutional neural network (BCNN) on both passive array and 1S1R array with different mapping schemes. Moreover, L2-norm gradient based learning and inference are also implemented on an RRAM network with both crossbar and coupled oscillators. For 3D CMOS-RRAM architecture, we also mapped the optimized SLFN algorithm. All the operations in learning and inference stages are implemented so that it can achieve the on-line learning. In addition, tensorized neural network (TNN) is mapped on both single-layer and multi-layer accelerators with different mapping scheme. All the machine learning algorithms mapped on XIMA and 3D CMOS-RRAM accelerator, we evaluate their performance in device, architecture and system levels. All the implementations show higher throughput, bandwidth and parallelism with better energy-efficiency.

7.2 Future Work

Based on the above works, there are a few potential future works for this thesis.

The first potential work is to validate some designs in the thesis. In this work, we explore the binary RRAM-crossbar as well as oscillator network in device and circuit level by simulation. However, the RRAM devices in the array are not fully uniform so that the mapping scheme may not be applied in a large crossbar array. In addition, sneak-path and thermal effect in the RRAM-crossbar will also occur the inaccuracy during computing. As for the distributed in-memory architecture, this work only simulates a few groups of control bus design. But in the real implementation, there will be thousands of control bus blocks so it is also necessary to optimize all these designs.

The second potential work is to realize neuromorphic computing by the RRAM network. This thesis only explores the artificial neural network implementations in the

RRAM-based architecture. All of these algorithms are originally designed for CMOS based computing systems such as CPU, GPU, CMOS-ASIC and FPGA. The RRAM based architecture can perform analog-value computing operations, which is widely applied in neuromorphic applications. For example, it is easier to implement spike-timing dependent plasticity (STDP) based synapse programming algorithm and also spiking neural networks (SNN) on RRAM compared to CMOS-ASIC or FPGA. It is possible to explore an RRAM-based neuromorphic computing system with simpler structure and better scalability than the state-of-arts.

Appendix A

Publication List

A.1 Book

1. Hao Yu, **Leibin Ni** and Yuhao Wang, “Non-Volatile In-Memory Computing by Spintronics”, *Synthesis Lectures on Emerging Engineering Technologies*, Morgan & Claypool Publisher, vol. 2, no. 7, pp. 1-161, December 2016.

A.2 Book chapter

1. Hao Yu, **Leibin Ni** and Hantao Huang, “Distributed In-Memory Computing on Binary Memristor-Crossbar for Machine Learning”, *Advances in Memristors, Memristive Devices and Systems*, Springer International Publishing, pp. 275-304, February 2017.

A.3 Journal

1. Zhongrui Wang, **Leibin Ni**, Jiaming Zhang, Joshi Saumil, Yunning Li, Peng Lin, Mingyi Rao, Rivu Midya, Can Li, Hao Jiang, Wenhao Song, Mark Barnell, Qing Wu, Hao Yu, Qiangfei Xia and J. Joshua Yang, “Small-scale Diffusive Memristor Synapses Array for Spiking Neural Network Based Unsupervised Learning”, *ACS Nano*, 2017. (to be submitted)
2. Hantao Huang, **Leibin Ni**, Kanwen Wang, Yuangang Wang and Hao Yu, “A Highly-parallel and Energy-efficient 3D Multi-layer CMOS-RRAM Accelerator for Tensorized Neural Network”, *IEEE Transactions on Nanotechnology (TNANO)*, 2017. (Accepted)

3. **Leibin Ni***, Zichuan Liu, Hao Yu and Rajiv V. Joshi, “An Energy-efficient Digital ReRAM-crossbar based CNN with Bitwise Parallelism”, *IEEE Journal of Exploratory Solid-State Computational Devices and Circuits (JxCDC)*, vol. 3, pp. 37-46, December 2017.
4. **Leibin Ni**, Hantao Huang, Zichuan Liu, Rajiv V. Joshi and Hao Yu, “Distributed In-memory Computing on Binary RRAM Crossbar”, *ACM Journal on Emerging Technologies in Computing (JETC)*, vol. 13, no. 3, pp. 36.1-36.18, April 2017.
5. Yuhao Wang*, **Leibin Ni**, Chip-Hong Chang and Hao Yu, “DW-AES: A Domain-wall Nanowire Based AES for High Throughput and Energy-efficient Data Encryption in Non-volatile Memory”, *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 11, no. 11, pp. 2426-2440, November 2016.
6. **Leibin Ni***, Sai Manoj P. D., Yang Song, Chenjie Gu and Hao Yu, “A Zonotoped Macromodeling for Eye-diagram Verification of High-speed I/O Links with Jitter and Parameter Variations”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol.35, no.6, pp. 1040-1051, June 2016.
7. Yuhao Wang*, Hao Yu, **Leibin Ni**, Mei Yan, Guangbin Huang, Chuliang Weng, Wei Yang and Junfeng Zhao, “An Energy-efficient Nonvolatile In-memory Computing Architecture for Extreme Learning Machine by Domain-wall Nanowire Devices”, *IEEE Transactions on Nanotechnology (TNANO)*, vol. 14, no. 6, pp. 998-1012, November 2015.

A.4 Conference

1. Hantao Huang, **Leibin Ni**, and Hao Yu, “TNN: An Energy-efficient Machine Learning Accelerator on 3D CMOS-RRAM for Tensorized Neural Network”, *IEEE International System-on-Chip Conference (SOCC)*, September 2017. (To appear)
2. **Leibin Ni**, Zichuan Liu, Wenhao Song, Jianhua Joshua Yang, Hao Yu, Kanwen Wang and Yuangang Wang, “An Energy-efficient and High-throughput Bitwise CNN on Sneak-path-free Digital ReRAM Crossbar”, *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, July 2017. (To appear)

3. Hantao Huang, **Leibin Ni**, Yuhao Wang, Hao Yu, Zongwei Wang, Yimao Cai and Ru Huang, “A 3D Multi-layer CMOS-RRAM Accelerator for Neural Network”, *IEEE International 3D System Integration Conference (3DIC)*, November 2016.
4. Hantao Huang, **Leibin Ni** and Hao Yu, A 3D Multi-layer CMOS-RRAM Accelerator for Multi-layer Machine Learning, *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, October 2016. (Invited)
5. **Leibin Ni***, Hantao Huang and Hao Yu, “A Memristor Network with Coupled Oscillator and Crossbar towards L2-norm based Machine Learning”, *ACM/IEEE International Symposium on Nanoscale Architectures (NANOARCH)*, July 2016.
6. Suman Deb, **Leibin Ni**, Hao Yu and Anupam Chattopadhyay, “Racetrack Memory-based Encoder/decoder for Low-power Interconnect Architectures”, *IEEE International Conference of Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2016. (Invited Special Session)
7. **Leibin Ni***, Hantao Huang and Hao Yu, “On-line Machine Learning Accelerator on Digital RRAM-Crossbar”, *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2016. (Invited Special Session)
8. Hao Yu, Hantao Huang and **Leibin Ni**, “Future brain-like computing with non-volatile memory device”, *China Semiconductor Technology International Conference (CSTIC)*, March 2016. (Invited Special Session)
9. **Leibin Ni***, Yuhao Wang, Hao Yu, Wei Yang, Chuliang Weng and Junfeng Zhao, “An Energy-efficient Matrix Multiplication Accelerator by Distributed In-memory Computing on Binary RRAM Crossbar”, *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, January 2016.
10. Yuhao Wang*, Xin Li, Hao Yu, **Leibin Ni**, Wei Yang, Chuliang Weng and Junfeng Zhao, “Optimizing Boolean Embedding Matrix for Compressive Sensing in RRAM Crossbar”, *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, August 2015.
11. Yuhao Wang*, Hantao Huang, **Leibin Ni**, Hao Yu, Mei Yan, Chuliang Weng, Wei Yang and Junfeng Zhao, “An Energy-efficient Non-volatile In-Memory Accelerator for Sparse-represented Face Recognition”, *ACM/IEEE Design Automation and Test Conference in Europe (DATE)*, March 2015.

A.5 Workshop/Poster

1. **Leibin Ni**, Zichuan Liu and Hao Yu, “A Binary Convolutional Neural Network on Digital ReRAM-Crossbar”, *ACM/IEEE Design Automation Conference (DAC)* (Work-in-progress session, poster), June 2017.
2. **Leibin Ni**, Sai Manoj P D, Yang Song and Hao Yu, “A Zonotoped Reachability Verification for High-speed I/O Links”, *International Workshop on Design Automation for Analog and Mixed-signal Circuits*, November 2014.
3. Sai Manoj P D, **Leibin Ni**, Hao Yu, Chenjie Gu and Cheng Zhuo, “A Zonotoped Macromodeling for Reachability Verification of Eye-diagram in High-speed I/O Links with Jitter”, *ACM/IEEE Design Automation Conference (DAC)* (Work-in-progress session, poster), June 2014.

Bibliography

- [1] H. Yu, L. Ni, and Y. Wang, *Non-Volatile In-Memory Computing by Spintronics*. Morgan & Claypool Publishers, 2016, vol. 2, no. 7.
- [2] H. Yu and Y. Wang, *Design exploration of emerging nano-scale non-volatile memory*. Springer, 2014.
- [3] D. Burger, J. R. Goodman, and A. Kägi, *Memory bandwidth limitations of future microprocessors*. ACM, 1996, vol. 24, no. 2.
- [4] S. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, T. Endoh, H. Ohno, and T. Hanyu, “Mtj-based nonvolatile logic-in-memory circuit, future prospects and issues,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 433–435.
- [5] S. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, H. Hasegawa, T. Endoh, H. Ohno, and T. Hanyu, “Fabrication of a nonvolatile full adder based on logic-in-memory architecture using magnetic tunnel junctions,” *Applied Physics Express*, vol. 1, no. 9, p. 091301, 2008.
- [6] W. H. Kautz, “Cellular logic-in-memory arrays,” *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 719–727, 1969.
- [7] H. Kimura, T. Hanyu, M. Kameyama, Y. Fujimori, T. Nakamura, and H. Takasu, “Complementary ferroelectric-capacitor logic for low-power logic-in-memory vlsi,” *IEEE Journal of Solid-State Circuits*, vol. 39, no. 6, pp. 919–926, 2004.
- [8] T. Hanyu, K. Teranishi, and M. Kameyama, “Multiple-valued logic-in-memory vlsi based on a floating-gate-mos pass-transistor network,” in *Solid-State Circuits Conference, 1998. Digest of Technical Papers. 1998 IEEE International*. IEEE, 1998, pp. 194–195.

- [9] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio, “The changing paradigm of data-intensive computing,” *Computer*, no. 1, pp. 26–34, 2009.
- [10] D. H. Wolpert, “The lack of a priori distinctions between learning algorithms,” *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.
- [11] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [12] K.-R. Müller, M. Tangermann, G. Dornhege, M. Krauledat, G. Curio, and B. Blankertz, “Machine learning for real-time single-trial eeg-analysis: from brain–computer interfacing to mental state monitoring,” *Journal of neuroscience methods*, vol. 167, no. 1, pp. 82–90, 2008.
- [13] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.
- [14] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: Theory and applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
- [15] A. Coates, A. Y. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *International conference on artificial intelligence and statistics*, 2011, pp. 215–223.
- [16] S. Park, M. Qazi, L.-S. Peh, and A. P. Chandrakasan, “40.4 fJ/bit/mm low-swing on-chip signaling with self-resetting logic repeaters embedded within a mesh NoC in 45nm SOI CMOS,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1637–1642.
- [17] V. Kumar, R. Sharma, E. Uzunlar, L. Zheng, R. Bashirullah, P. Kohl, M. S. Bakir, and A. Naeemi, “Airgap interconnects: Modeling, optimization, and benchmarking for backplane, PCB, and interposer applications,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 4, no. 8, pp. 1335–1346, 2014.
- [18] Y. Wang, H. Yu, L. Ni, G.-B. Huang, M. Yan, C. Weng, W. Yang, and J. Zhao, “An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices,” *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 998–1012, 2015.

- [19] S. Matsunaga, J. Hayakawa, S. Ikeda, K. Miura, T. Endoh, H. Ohno, and T. Hanyu, "MTJ-based nonvolatile logic-in-memory circuit, future prospects and issues," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 433–435.
- [20] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2237–2251, 2010.
- [21] 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. 1, pp. 389–395, 2011.
- [22] L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
- [23] S. R. Williams, "How we found the missing memristor," *IEEE Spectrum*, vol. 45, no. 12, pp. 28–35, 2008.
- [24] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.
- [25] Y. Shang, W. Fei, and H. Yu, "Analysis and modeling of internal state variables for dynamic effects of nonvolatile memory devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 9, pp. 1906–1918, 2012.
- [26] W. Fei, H. Yu, W. Zhang, and K. S. Yeo, "Design exploration of hybrid CMOS and memristor circuit by new modified nodal analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1012–1025, 2012.
- [27] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu, and J. Yang, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, pp. 66.1–66.6.
- [28] Y. Kim, Y. Zhang, and P. Li, "A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning," in *International SOC Conference (SOCC)*. IEEE, 2012, pp. 328–333.
- [29] W. Lu, K.-H. Kim, T. Chang, and S. Gaba, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2011, pp. 217–223.

- [30] H. Yu, L. Ni, and H. Huang, *Distributed In-Memory Computing on Binary Memristor-Crossbar for Machine Learning*. Springer, 2017, pp. 275–304.
- [31] L. Chua, “Memristor-the missing circuit element,” *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.
- [32] D. B. Strukov and R. S. Williams, “Exponential ionic drift: Fast switching and low volatility of thin-film memristors,” *Applied Physics A*, vol. 94, no. 3, pp. 515–519, 2009.
- [33] Y. N. Joglekar and S. J. Wolf, “The elusive memristor: properties of basic electrical circuits,” *European Journal of Physics*, vol. 30, no. 4, p. 661, 2009.
- [34] Z. Biolek, D. Biolek, and V. Biolkova, “Spice model of memristor with nonlinear dopant drift,” *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.
- [35] Z. Wang, S. Joshi, S. E. Savel’ev, H. Jiang, R. Midya, P. Lin, M. Hu, N. Ge, J. P. Strachan, Z. Li, Q. Wu, M. Barnell, G.-L. Li, H. L. Xin, R. S. Williams, Q. Xia, and J. J. Yang, “Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing,” *Nature Materials*, vol. 16, no. 1, pp. 101–108, 2017.
- [36] R. Midya, Z. Wang, J. Zhang, S. E. Savel’ev, C. Li, M. Rao, M. H. Jang, S. Joshi, H. Jiang, P. Lin, K. Norris, N. Ge, Q. Wu, M. Barnell, Z. Li, H. L. Xin, R. S. Williams, Q. Xia, and J. J. Yang, “Anatomy of Ag/Hafnia-based selectors with 10^{10} nonlinearity,” *Advanced Materials*, vol. 29, no. 12, 2017.
- [37] B. Govoreanu, G. S. Kar, Y.-Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. J. Wouters, J. A. Kittl, and M. Jurczak, “ $10 \times 10 \text{ nm}^2$ Hf/HfO_x crossbar resistive RAM with excellent performance, reliability and low-energy operation,” in *International Electron Devices Meeting (IEDM)*, 2011, pp. 31.6.1–31.6.4.
- [38] J. Kang, B. Gao, B. Chen, P. Huang, F. F. Zhang, Y. X. Deng, L. F. Liu, X. Y. Liu, H. Y. Chen, Z. Jiang, S. M. Yu, and H. S. P. Wong, “3D RRAM: Design and optimization,” in *IEEE Conference on International Solid-State and Integrated Circuit Technology (ICSICT)*, 2014, pp. 1–4.
- [39] D. Fan, M. Sharad, and K. Roy, “Design and synthesis of ultralow energy spin-memristor threshold logic,” *IEEE Transactions on Nanotechnology*, vol. 13, no. 3, pp. 574–583, 2014.

- [40] P. Gu, B. Li, T. Tang, S. Yu, Y. Cao, Y. Wang, and H. Yang, “Technological exploration of RRAM crossbar array for matrix-vector multiplication,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2015, pp. 106–111.
- [41] T. Srimani, B. Manna, A. K. Mukhopadhyay, K. Roy, and M. Sharad, “Energy efficient and high performance current-mode neural network circuit using memristors and digitally assisted analog CMOS neurons,” *arXiv preprint arXiv:1511.09085*, 2015.
- [42] Y. Wang, H. Yu, and W. Zhang, “Nonvolatile CBRAM-crossbar-based 3-D-integrated hybrid memory for data retention,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 5, pp. 957–970, 2014.
- [43] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, “Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication,” in *Design Automation Conference (DAC)*, 2016, pp. 19.1–19.6.
- [44] L. Xia, T. Tang, W. Huangfu, M. Cheng, X. Yin, B. Li, Y. Wang, and H. Yang, “Switched by input: Power efficient structure for RRAM-based convolutional neural network,” in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, pp. 125.1–125.6.
- [45] L. Ni, Y. Wang, H. Yu, W. Yang, C. Weng, and J. Zhao, “An energy-efficient matrix multiplication accelerator by distributed in-memory computing on binary RRAM crossbar,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 280–285.
- [46] L. Ni, H. Huang, and H. Yu, “On-line machine learning accelerator on digital RRAM-crossbar,” in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 113–116.
- [47] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, “Binary convolutional neural network on RRAM,” in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 782–787.
- [48] B. Yan, A. M. Mahmoud, J. J. Yang, Q. Wu, Y. Chen, and H. H. Li, “A neuromorphic ASIC design using one-selector-one-memristor crossbar,” in *International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 1390–1393.

- [49] T. C. Jackson, A. A. Sharma, J. A. Bain, J. A. Weldon, and L. Pileggi, "Oscillatory neural networks based on tmo nano-oscillators and multi-level rram cells," *IEEE journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 230–241, 2015.
- [50] A. A. Sharma, Y. Li, M. Skowronski, J. A. Bain, and J. A. Weldon, "High-frequency TaOx-based compact oscillators," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3857–3862, Nov 2015.
- [51] P. Maffezzoni, L. Daniel, N. Shukla, S. Datta, A. Raychowdhury, and V. Narayanan, "Modelling hysteresis in vanadium dioxide oscillators," *Electronics Letters*, vol. 51, no. 11, pp. 819–820, 2015.
- [52] S. S. Haykin, *Neural networks and learning machines*. Pearson Education Upper Saddle River, 2009, vol. 3.
- [53] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1-3, pp. 37–52, 1987.
- [54] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.
- [55] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.
- [56] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [57] N. J. Higham, "Cholesky factorization," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 2, pp. 251–254, 2009.
- [58] A. Krishnamoorthy and D. Menon, "Matrix inversion using cholesky decomposition," *arXiv preprint arXiv:1111.4144*, 2011.
- [59] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International Conference on Artificial Neural Networks (ICANN)*. Springer, 2014, pp. 281–290.
- [60] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
- [61] L. Zhang and P. Suganthan, "A comprehensive evaluation of random vector functional link networks," *Information Sciences*, vol. 367, pp. 1094–1105, 2016.

- [62] K. Bache and M. Lichman, “UCI machine learning repository,” 2013. [Online]. Available: <http://archive.ics.uci.edu/ml>
- [63] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [64] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face recognition: A convolutional neural-network approach,” *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.
- [65] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.
- [66] M. Courbariaux, Y. Bengio, and J.-P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 3123–3131.
- [67] Z. Liu, Y. Li, F. Ren, and H. Yu, “A binary convolutional encoder-decoder network for real-time natural scene text processing,” *arXiv preprint arXiv:1612.03630*, 2016.
- [68] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [69] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, “Going deeper with embedded FPGA platform for convolutional neural network,” in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016, pp. 26–35.
- [70] D. C. Plaut, “Experiments on learning by back propagation,” Carnegie Mellon University, Tech. Rep., 1986.
- [71] I. Hubara, D. Soudry, and R. E. Yaniv, “Binarized neural networks,” *arXiv preprint arXiv:1602.02505*, 2016.
- [72] A. Davis and I. Arel, “Low-rank approximations for conditional feedforward computation in deep neural networks,” *arXiv preprint arXiv:1312.4461*, 2013.
- [73] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, “Compressing deep neural networks using a rank-constrained topology,” in *INTERSPEECH*, 2015, pp. 1473–1477.

- [74] A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 442–450.
- [75] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
- [76] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," *arXiv preprint arXiv:1403.2048*, 2014.
- [77] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural network design*. PWS publishing company Boston, 1996, vol. 20.
- [78] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*, 2007, pp. 153–160.
- [79] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multi-layer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809–821, 2016.
- [80] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 210–227, 2009.
- [81] H. Lee, P. Chen, T. Wu, Y. Chen, C. Wang, P. Tzeng, C. Lin, F. Chen, C. Lien, and M.-J. Tsai, "Low power and high speed bipolar switching with a thin reactive Ti buffer layer in robust HfO₂ based RRAM," in *International Electron Devices Meeting (IEDM)*. IEEE, 2008, pp. 1–4.
- [82] P. N. Singh, A. Kumar, C. Debnath, and R. Malik, "20mW, 125Msps, 10bit pipelined ADC in 65nm standard digital CMOS process," in *Custom Integrated Circuits Conference (CICC)*. IEEE, 2007, pp. 189–192.
- [83] T. Tan and Z. Sun, "CASIA-FingerprintV5," 2010. [Online]. Available: <http://biometrics.idealtest.org/>
- [84] D. M. Chiarulli, B. Jennings, Y. Fang, A. Seel, and S. P. Levitan, "A computational primitive for convolution based on coupled oscillator arrays," in *Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2015, pp. 125–130.

- [85] K. Yogendra, D. Fan, Y. Shim, M. Koo, and K. Roy, "Computing with coupled spin torque nano oscillators," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016, pp. 312–317.
- [86] R. O. Topaloglu, *More than moore technologies for next generation computer design*. Springer, 2015.
- [87] Y. Y. Liauw, Z. Zhang, W. Kim, A. El Gamal, and S. S. Wong, "Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory," in *International Solid-State Circuits Conference (ISSCC)*. IEEE, 2012, pp. 406–408.
- [88] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *IEEE DATE*, 2015, pp. 1543–1546.
- [89] Y.-C. Chen, W. Wang, H. Li, and W. Zhang, "Non-volatile 3D stacking RRAM-based FPGA," in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 367–372.
- [90] L. Ni, Z. Liu, H. Yu, and R. Joshi, "An energy-efficient digital reram-crossbar based cnn with bitwise parallelism," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 3, no. 1, pp. 1–10, 2017.
- [91] S. Yu, Z. Li, P.-Y. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian, "Binary neural network with 16 Mb RRAM macro chip for classification and online training," in *International Electron Devices Meeting*, 2016, pp. 16.2.1–16.2.4.
- [92] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 525–542.
- [93] L. Ni, H. Huang, and H. Yu, "A memristor network with coupled oscillator and crossbar towards L2-norm based machine learning," in *International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2016, pp. 179–184.
- [94] H. Huang, L. Ni, Y. Wang, H. Yu, Z. Wang, Y. Cai, and R. Huang, "A 3D multi-layer CMOS-RRAM accelerator for nueral network," in *International Conference on 3D System Integration (3DIC)*. IEEE, 2016.

- [95] P.-Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, "Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 854–859.
- [96] L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu, "Distributed in-memory computing on binary RRAM crossbar," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 36.1–36.18, 2017.
- [97] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep., 2007.
- [98] A. Vedaldi and K. Lenc, "Matconvnet: Convolutional neural networks for Matlab," in *International Conference on Multimedia*. ACM, 2015, pp. 689–692.
- [99] B. Goll and H. Zimmermann, "A 65nm CMOS comparator with modified latch to achieve 7GHz/1.3mW at 1.2V and 700MHz/47 μ W at 0.6V," in *International Solid-State Circuits Conference (ISSCC)*. IEEE, 2009, pp. 328–329.
- [100] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 2015, pp. 161–170.
- [101] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 262–263.
- [102] G. E. Hinton and R. R. Salakhutdinov, "Replicated softmax: An undirected topic model," in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1607–1614.
- [103] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist>
- [104] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," 2014. [Online]. Available: <http://www.cs.toronto.edu/~kriz/cifar.html>
- [105] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

- [106] P. Franzon, E. Rotenberg, J. Tuck, W. R. Davis, H. Zhou, J. Schabel, Z. Zhang, J. B. Dwiel, E. Forbes, J. Huh *et al.*, “Computing in 3D,” in *Custom Integrated Circuits Conference (CICC)*. IEEE, 2015, pp. 1–6.
- [107] D. H. Kim, K. Athikulwongse, and S. K. Lim, “Study of through-silicon-via impact on the 3-D stacked IC layout,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 862–874, 2013.
- [108] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” University of Toronto, Tech. Rep., 2009.
- [109] “GPU specs.” [Online]. Available: <http://www.nvidia.com/object/workstation-solutions.html>
- [110] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, “CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory,” in *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 2012, pp. 33–38.
- [111] I. V. Oseledets and S. Dolgov, “Solution of linear systems and matrix inversion in the tt-format,” *SIAM Journal on Scientific Computing*, vol. 34, no. 5, pp. A2718–A2739, 2012.
- [112] J. Xue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” in *INTERSPEECH*, 2013, pp. 2365–2369.
- [113] “CPU Xeon-X5690specs,” accessed: 2017-03-30. [Online]. Available: http://ark.intel.com/products/52576/Intel-Xeon-Processor-X5690-12M-Cache-3_46-GHz-6_40-GTs-Intel-QPI
- [114] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.