Algorithm and Hardware Co-design

for Learning On-a-chip

by

Zihan Xu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved August 2017 by the
Graduate Supervisory Committee:

Yu Cao, Chair
Chaitali Chakrabarti
Jae-sun Seo
Shimeng Yu

ARIZONA STATE UNIVERSITY

December 2017

ABSTRACT

Machine learning technology has made a lot of incredible achievements in recent years. It has rivalled or exceeded human performance in many intellectual tasks including image recognition, face detection and the Go game. Many machine learning algorithms require huge amount of computation such as in multiplication of large matrices. As silicon technology has scaled to sub-14nm regime, simply scaling down the device cannot provide enough speed-up any more. New device technologies and system architectures are needed to improve the computing capacity. Designing specific hardware for machine learning is highly in demand. Efforts need to be made on a joint design and optimization of both hardware and algorithm.

For machine learning acceleration, traditional SRAM and DRAM based system suffer from low capacity, high latency, and high standby power. Instead, emerging memories, such as Phase Change Random Access Memory (PRAM), Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM), and Resistive Random Access Memory (RRAM), are promising candidates providing low standby power, high data density, fast access and excellent scalability. This dissertation proposes a hierarchical memory modeling framework and models PRAM and STT-MRAM in four different levels of abstraction. With the proposed models, various simulations are conducted to investigate the performance, optimization, variability, reliability, and scalability.

Emerging memory devices such as RRAM can work as a 2-D crosspoint array to speed up the multiplication and accumulation in machine learning algorithms. This dissertation proposes a new parallel programming scheme to achieve in-memory learning with RRAM crosspoint array. The programming circuitry is designed and simulated in

TSMC 65nm technology showing 900X speedup for the dictionary learning task compared to the CPU performance.

From the algorithm perspective, inspired by the high accuracy and low power of the brain, this dissertation proposes a bio-plausible feedforward inhibition spiking neural network with Spike-Rate-Dependent-Plasticity (SRDP) learning rule. It achieves more than 95% accuracy on the MNIST dataset, which is comparable to the sparse coding algorithm, but requires far fewer number of computations. The role of inhibition in this network is systematically studied and shown to improve the hardware efficiency in learning.

ACKNOWLEDGMENTS

Foremost, I would like to sincerely thank my advisor Dr. Yu Cao for the support of my Ph.D research. He has guided me the professional way to do research patiently. He has helped me solve difficult prolems in my research. His motivation and enthusiasm always inspire me to work harder and pursue the truth. His knowledge and vision has directed me towards a great career. I could not have imagined having a better advisor for my Ph.D study.

Besides my advisor, I would like to thank the reset of my dissertation committee: Dr. Chaitali Chakrabarti, Dr. Jae-sun Seo, and Dr. Shimeng Yu. They have helped me a lot through my projects with their professional knowledge and attitude. They have given me a broad view and experience of different research fields.

I also want to thank my fellows in Arizona State University for the help, discussion, and support: Dr. Chengen Yang, Dr. Ketul Sutaria, Dr. Naveen Suda, Dr. Jyothi Velamala, Dr. Saurabh Sinha, Pei An, Venkatesa Sarma Ravi, Abinash Mohanty, Devyani Patra, Srivatsava Gorthy, Xiaocong Du, Zheng Li, Shihui Yin, Deepak Kadetotad, Xiaoyang Mi, Pai-Yu Chen, Dr. Ligang Gao, Runchen Fang, Zhiwei Li, Rui Liu, Manqing Mao, Dr. Wenhao Chen, Dr. Binbin Lin, Ming Tu.

I would like to thank all the faculty and staff in Arizona State University for the great Electrical Engineering program.

Last but not the least, I want to thank my parents for their unreserved support throughout my life.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

1. Introduction

1.1. Background

Machine learning technology, especially deep learning, has a lot of incredible achievements in recent years. Machine learning based products have been integrated into our daily life, such as spam filter for email, recommendation system on ecommerce websites, and automatic face detection when taking photos with the smart phones. It has also rivalled or exceeded human performance in many intellectual tasks including image recognition, face detection and the Go game. In the image recognition task of 1,000 categories (ImageNet Large Scale Visual Recognition Challenge), the state-of-the-art algorithm achieves 2.25% top-5 error rate (ILSVRC 2017), which is better than the human performance of 5.1% top-5 error rate (Karpathy 2014).

Machine learning is the science of getting computers to learn, without being explicitly programmed. The computers do not copy human's knowledge directly, but learn from data with human defined rules. Machine learning is a very broad concept, and this dissertation only focuses on neural networks. A deep neural network model usually needs to learn millions of parameters provided millions of data (LeCun et al. 2015, 521:436-444). Thus, machine learning algorithms require huge amount of computations and data movement.

The current success of machine learning can be attributed to three aspects. First, the computational capability of modern computers have been increasing rapidly. Especially the development of Graphics Processing Unit (GPU) makes large scale parallel computation possible and efficient (Raina et al. 2009, 873-880). And deep learning models, especially neural networks, are featured by highly parallel computation of matrix multiplication and convolution. With GPUs, people can train a deep neural network much faster than before. This not only speeds up the developing and tuning cycle of the algorithms, but also makes high complexity models possible. Second, as Internet spreads all over the world, there are huge amount of data generated by everyone every day, such as articles, pictures and search history. These “big data” on the Internet can be easily accessed and collected. And large amount of data is crucial to the training of any machine learning algorithm. Third, new algorithms and techniques were proposed to improve learning when simply increasing the model complexity cannot improve the performance due to overfitting, e.g. regularization, dropout (Srivastava et al. 2014, 873-880), and residual learning (He et al. 2015). Due to the reasons above, the tremendous growth of machine learning depends on the development of both hardware and algorithm.

The demand for faster computing is increasing every day. The semiconductor hardware industry has been following Moore’s law due to a variety of benefits of scaling (Moore 1965, Moore 1975). The number of transistors on a chip doubles every two years. The performance of a chip doubles every 18 months due to more and faster transistors. And the cost of chips becomes more affordable to more people. Although Moore’s law came from Gordon Moore's observation rather than any scientific or engineering theory, it has been very successful for over 50 years. However, as the silicon technology has scaled to sub-14nm regime, simply scaling down the device cannot provide enough speed-up any more. New device technologies and system architectures are needed to improve the computing capacity and continue Moore’s law. Moreover, specific hardware for machine learning acceleration is highly in demand.

Based on different applications of machine learning, there will be different specifications and hardware choices. Machine learning algorithms usually have two phases, the training phase and the inference phase. During training, the model computes the forward path and backward path to update the parameters. During inference, the model parameters are fixed, and the model only computes the forward path to get the output for a given input. Basically, training is to obtain the model while inference is to use the model. Thus, training requires more computing and memory accesses. Designing hardware for training is more challenging than inference. Also, the hardware for inference can be very specific to the model to achieve better performance, while the hardware for training needs to be more general since the model to be trained often changes. From the perspective of where the machine learning models are used, they can be used in cloud servers or on edge devices. In the cloud, complex deep learning models are used to do training or inference on "big data". Speed is important to shorten the model development cycle or reduce the query latency. On the edge, more and more machine learning applications like voice recognition, machine translation and personalized health care will be implemented on the mobile devices. Therefore, application in the cloud may prefer hardware optimized for high speed, while application on the edge may prefer hardware optimized for low power.

There are different hardware choices based on different specifications. CPU is good for light load due to its high frequency, low power and high flexibility. GPU is now widely used to train deep learning models, which is much faster than CPU due to its Single Instruction Multiple Data (SIMD) operation and high bandwidth memory. However, GPU is not yet specialized for machine learning and it consumes a lot of power. Recently, more attention is paid to design of Application Specific Integrated Circuit (ASIC) for machine learning (Misra 2016, 74:239-255). It has the promise to outperform GPU in terms of speed and power. But ASIC design usually needs a long design and verification cycle, and the software ecosystem needs to follow up quickly. Another great option is the Field Programmable Gate Array (FPGA) (Misra and Saha 2016, 74:239-255). The computing capacity and memory capacity of FPGAs are improving quickly. And the development cycle of FPGA is much shorter than ASIC. These features make FPGA an excellent choice for certain fast changing applications.

The above acceleration options are mostly based on parallel computation with SIMD. Most efforts are made to include more parallel Processing Elements (PE) and increase the memory bandwidth, so that the computing capacity and the data movement capacity are improved together. However, in these approaches, data storage and computing are separated. Data need to be transferred from memory to the registers in PE to be processed. Since it's quite possible to increase the number of parallel PEs to speed up computing, the memory bandwidth will eventually become the bottleneck. Therefore, a more advanced idea is to do the computation inside the memory. This requires a new type of memory device that enables computing capability.

Learning on-a-chip requires not only hardware design and implementation, but also optimization of the algorithms to be easier to implement on hardware. One approach is to reduce the number of bits of data including parameters and variables. In the current computer system, data are usually represented by 32 bits, such as single-precision floating-point number and integer type. Many researches have shown that machine learning algorithms do not actually need 32-bit numbers (Gupta et al. 2015, Baldassi et al. 2016, 93:052313, Merolla et al. 2016). Reducing the number of bits will benefit computation, data movement and memory storage for hardware implementation.

Similarly in human brain, the basic elements, the neurons and synapses, do not have high numerical precision, but the brain has excellent performance and extreme low power in the cognitive tasks. It manages a variety of tasks with such a small volume. Thus biology and neuroscience provide motivation to improve the current learning algorithms and their hardware implementation efficiency.

1.2. Contributions

As silicon technology scales down, traditional CMOS-based memory such as SRAM and DRAM will suffer from high standby power consumption. They may no longer be the technology of choice for machine learning acceleration. Instead, emerging memories, such as Phase Change Random Access Memory (PRAM) (Burr et al. 2010, 28:223-262, Wong et al. 2010, 98 :2201-2227), Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) (Wang et al. 2008, 44:2479-2482, Sharad et al. 2012, 11:843-853), and Resistive Random Access Memory (RRAM) (Wong et al. 2012, 100:1951-1970, Jo et al. 2010, 10:1297-1301, Yu et al. 2013, 25:1774-1779), are promising candidates for the next generation non-volatile memory. They provide low standby power, high data density, fast access and excellent scalability. They are also very promising to be integrated into the machine learning chip with CMOS technology. More importantly, they have the potential to do computation. Therefore, they are great candidates in the machine learning hardware. Modeling of these emerging memory devices is essential to the design. This dissertation proposes a hierarchical memory modeling framework and models PRAM and STT-MRAM in four different levels of abstraction (Xu et al. 2012, Xu et al. 2013, Xu et al. 2014, 102:76-

81). With the proposed models, various simulations are conducted to investigate the performance, optimization, variability, reliability, and scalability of these two memories.

Emerging non-volatile memory devices such as RRAM can work as a 2-D crosspoint array, which is a promising structure for matrix-based machine learning algorithms. It can do parallel computation of multiplication and accumulation based on current-voltage relationship. Thus, it reduces the need for data movement since the computation happens in the memory itself. This architecture is different from the mainstream Von-Neumann architecture, and is more suitable for data intensive applications like machine learning algorithms. How to integrate the non-volatile memory array with the computing system is a very challenging problem. It requires proper circuit and system design to achieve high speed and low power. This dissertation proposes a new parallel programming scheme to achieve in-memory learning with RRAM crosspoint array (Xu et al. 2014, 41:126-133). Based on the scheme, the programming circuitry is designed and simulated in TSMC 65nm technology. The proposed parallel programming scheme can speed up the dictionary learning task significantly compared to the CPU.

Another approach to optimize the algorithms is to learn from the biological nervous system. Inspired by the brain, which has high accuracy, low power and high hardware efficiency, many learning models were developed with spiking neurons (Cao et al. 2015, 113:54-66, Diehl et al. 2015). This dissertation proposes a bio-plausible feedforward inhibition spiking neural network with Spike-Rate-Dependent-Plasticity (SRDP) learning rule (Xu et al. 2016, Xu et al. 2017). It achieves more than 95% accuracy on the standard benchmark MNIST dataset, which is comparable to the sparse coding algorithm, but needs much less number of computations. The inhibition in this network plays an important role in improving the hardware efficiency of learning.

The rest of the dissertation is organized as follows. Chapter 2 reviews related works on machine learning algorithm optimization with low numerical precision, and machine learning hardware design including ASIC and FPGA. Chapter 3 focuses on emerging memory devices. Modeling and design exploration are conducted for Phase Change Random Access Memory (PRAM) and Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM). Resistive Random Access Memory (RRAM) is also reviewed in this chapter to be used in next chapter. Chapter 4 designs the new parallel programming circuitry for learning on an RRAM crosspoint array. In Chapter 5, existing learning algorithms are firstly reviewed, and a new bio-plausible learning algorithm of feedforward inhibition spiking neural network is proposed and demonstrated with the standard benchmark MNIST dataset. Finally, Chapter 6 proposes the future works and Chapter 7 concludes this dissertation.

2. Related Work

This chapter reviews related works on machine learning algorithm optimization with low numerical precision, and machine learning hardware design including ASIC, FPGA and emerging non-volatile memory array.

2.1. Machine Learning Algorithm Optimization

To optimize machine learning algorithms for hardware implementation, many efforts were spent on reducing the numerical precision due to its benefits to computation, data movement and memory storage. Many works focused on the training of models. Hollis et al. (1990, 2:363-373) studied the effects of precision constraints on the backpropagation in neural network training. Fixed point arithmetic was used in the analog circuits to implement the backpropagation. Holt et al. (1993, 42:281-290) theoretically analyzed the finite precision error of the computations in training Multi-Layer Perceptron (MLP) for hardware implementation. Plagianakos and Vrahatis (1999) used the differential evolution strategy to train neural networks of 3-bit integer weights on simple datasets. For more complex deep learning algorithms, Baldassi et al. (2016, 93:052313) did the theoretically analysis for the possibility that learning may need only few bits of synaptic precision.

Gupta et al. (2015) proposed the stochastic rounding method to train deep learning models with 16-bit fixed-point number. It achieved 25.4% testing error on CIFAR-10 dataset (Krizhevsky and Hinton 2009), which is very close to the floating-point model baseline. Gysel et al. (2016) managed to condense CaffeNet model to 8-bit fixed-point representation using quantization and fine-tuning method. Lin et al. (2016) further proposed 3 fine-tuning methods and did a systematical study. They studied the effect of low precision weights and activations separately. They also showed different layers in the model have different effect of low precision on the accuracy.

More aggressively, there are many works trying to train neural networks with binary weights and activations. Baldassi (2009, 136:902-916) studied generalization learning in a perceptron with binary synapses. Kim and Smaragdis (2016) proposed the weight compression and noisy backpropagation method to train neural networks with binary weights and activations. They achieved 1.33% error rate on the MNIST dataset (LeCun et al. 1998, 86:2278-2324). Courbariaux has a series of three papers (Courbariaux et al. 2014, Courbariaux et al. 2015, Courbariaux et al. 2016) on the binary neural networks and studied the more difficult CIFAR-10 dataset. They used the method of propagating gradients through discretization and batch normalization in training, and achieved 9.9% error rate on CIFAR-10 dataset with the VGG model baseline (Simonyan and Zisserman 2014). Rastegari et al. (2016) proposed a new method to project the binary weights based on mathematical derivation and applied this on AlexNet (Krizhevsky 2012) on the more complex ImageNet dataset. Compared to the 80.2% top-5 accuracy of full-precision AlexNet, they achieved 69.2% accuracy with binary weight and binary activation model, and achieved 79.4% accuracy with binary weight only model. Zhou et al. (2016) studied various number of bits of weights and activations in the AlexNet model, and achieved 43.6% accuracy. Merolla et al. (2016) not only studied the weight binarization but also other non-linear distortions during training. They showed that deep neural networks are robust to them with experiments on CIFAR-10 and ImageNet dataset. Stromatias et al. (2015) also showed the robustness of spiking deep belief networks to noise and reduced bit precision. With all these efforts above, the binary neural networks training can achieve the same accuracy as the full-precision counterpart on some datasets, but still cannot compete on the complex ImageNet dataset. Li et al. (2016) proposed the ternary weight networks and applied on the ResNet (He et al. 2016). They showed 84.2% accuracy on ImageNet dataset compared to the 86.76% accuracy of the full-precision model. Other experiments also showed that the accuracy of the ternary weight networks is almost the same as the full-precision counterparts.

Besides the low precision models, Iandola et al. (2016) tried to reduce the size of neural networks for easier implementation on the embedded hardware. They proposed a small CNN architecture called SqueezeNet. It achieved AlexNet-level accuracy on ImageNet dataset with 50x fewer parameters. For inference purpose only, Han et al. (2015) proposed the deep compression method including pruning, trained quantization and Huffman coding. They managed to reduce the memory storage of VGG-16 model by 49X with no loss of accuracy.

2.2. Machine Learning Hardware Design

Many efforts were made to design machine learning acceleration hardware including ASIC and FPGA. Chen et al. proposed a series of ASIC designs called “DianNao family” (Chen et al. 2016, 59:105-112, Chen et al. 2014, 49:269-284, Luo et al. 2017, 66:73-88, Du et al. 2015, 43:92-104). They specially emphasized the impact of the memory on machine learning accelerator design, performance and energy. Among the DianNao family, DaDianNao was designed for neural networks and achieved peak performance of 5585 giga operations per second (GOPS) with peak power of 15.97 W in 28 nm technology. This is a 450X speedup over GPU with 150X less energy comsumption. It applied loop tiling and SIMD to minimize the memory access. ShiDianNao was designed for convolutional neural networks (CNN) and achieved 194 GOPS with 0.32 W in 65 nm technology. Since the storage size of convolution kernels is not very large, this design stored the parameters in SRAM rather than DRAM, thus saving a lot of time and energy in memory access. A group from MIT proposed an energy-efficient reconfigurable ASIC design for CNN (Chen et al. 2016, 52:127-138). The design adopted special dataflow according to the behavior of convolution thus maximized the data reuse. The chip implemented in 65 nm technology achieved 34.7 images per second of the five convolutional layers inference in AlexNet with power of 278 mW. Google recently revealed more details about their Tensor Processing Unit (TPU) (Jouppi et al. 2017). It achieved 180 tera floating point operations per second (TFLOPS) with 4 chips on the board. The 2D systolic array is adopted in matrix multiplication to improve the throughput.

Apart from these designs accelerating the arithmetic computations in neural networks, IBM designed the chip called TrueNorth implementing the spiking neural networks (Merolla et al. 2014, 345:668-673). They converted the convolutional neural network to the spiking version and implemented on the TrueNorth chip. They achieved more than 1000 images per second for inference.

To further speed up the inference, Han et al. (2016) designed an ASIC of efficient inference engine (EIE) on compressed deep neural networks as a follow-up work of the deep compression work. It achieved 102 GOPS on compressed models and 3 TOPS on uncompressed models.

There are also many works using FPGA as the machine learning accelerator. Farabet et al. (2011) designed a runtime reconfigurable dataflow processor and developed the compiler to map CNN algorithms on the hardware design. They demonstrated the street scene understanding application and achieved 160 peak GOPS with only 10 W power. Gokhale et al. (2014) designed a mobile coprocessor for CNN and achieved 240 GOPS. Zhang et al. (2015) used a roofline model to explore the design space of the computation throughput and memory bandwidth, and achieved 61.62 GFLOPS under 100 MHz working frequency for CNN. Suda et al. (2016) proposed a throughput-optimized OpenCL based design on FPGA for CNN. They achieved 136.5 GOPS for the convolution operation. Qiu et al. (2016) implemented VGG-16 model on FPGA using RTL design. The data precision is quantized to 8 bits and even 4 bits with only 0.4% accuracy loss. They achieved 187.8 GOPS for convolution and 137 GOPS for the full model. Ma et al. (2016) proposed a scalable and modular RTL compiler of CNN. They optimized the CNN operations in RTL and developed a compiler to map the CNN models to the RTL module. So, it integrates the flexibility of high level synthesis (HLS) and the finer level optimization of RTL. They achieved 114.5 GOPS for the AlexNet model. Wei et al. (2017) implemented the systolic array architecture on FPGA and achieved impressive 461 GFLOPS for floating point data type and 1.2 TOPS for 8-16 bits fixed point data. Besides the above efforts to reduce the data precision, there are a few works to implement the binary neural networks on FPGA utilizing the flexibility of FPGA (Nurvitadhi et al. 2016, Fraser et al. 2017, Zhao et al. 2017). They have achieved peak throughput from 207.8 GOPS up to 14.8 TOPS.

There are also some other works using the emerging non-volatile memory array. Park et al. (2012) firstly demonstrated the use of a 1k-bit RRAM crosspoint array to speed up machine learning algorithms. Garbin et al. (2014) proposed a spike-based implementation of CNN using binary RRAM devices and achieved 94% accuracy on MNIST dataset. Xia et al. (2015) did a thorough technological exploration RRAM crosspoint array for matrix-vector multiplication. They studied the non-ideal factors of the device, variations, and parasitics, and analyzed how to achieve a better trade-off between performance, energy and reliability. Prezioso et al. (2015) did an experimental implementation of RRAM crosspoint array for a single layer perceptron. They demonstrated the training of the model on the RRAM crosspoint array and achieved a good classification result on a small task. Hu et al. (2016) developed the Dot-Product Engine with RRAM crosspoint array and showed 1,000X to 10,000X more speed-energy efficiency product than state-of-the-art ASIC design. Gokmen and Vlasov (2016, 10) applied stochastic computing and stochastic update rule on the RRAM crosspoint array. They achieved 98% accuracy on MNIST dataset by training a CNN model. A full update cycle of an array performed using 1 ns pulses can be completed in 20 ns with 0.28 W power. Chi et al. (2016) designed the entire system and software-hardware interface to use RRAM crosspoint array as matrix-vector multiplication accelerator called PRIME. It shows 1,596X to 73,237X speedup and 335X to 138,984X power consumption reduction over CPU.

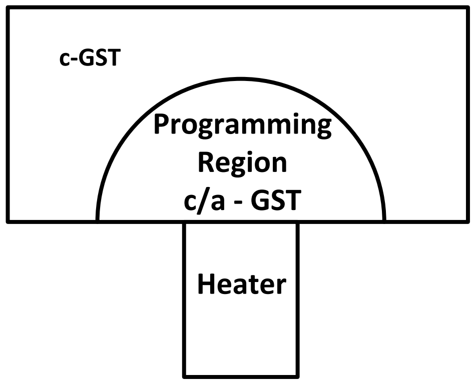
3. Emerging Memory Modeling

In this chapter, a hierarchical framework is proposed for the emerging memory modeling. Phase Change Random Access Memory (PRAM) and Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) are modeled explicitly. With the proposed compact models, early stage design benchmarking is performed for these two types of memory. In addition, Resistive Random Access Memory (RRAM) is reviewed for the completion of the scope and will be used in the later work.

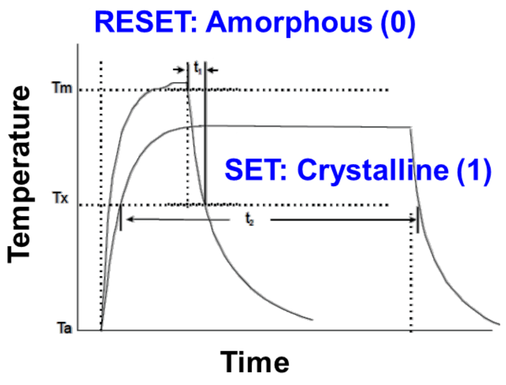
3.1. Introduction

As Silicon technology is scaling down toward the 10nm regime, CMOS-based memory devices such as SRAM and DRAM suffer from high standby power consumption, so that they may no longer be the technology of choice. Instead, emerging memories, such as Phase Change Random Access Memory (PRAM) and Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM), are promising candidates for the next generation non-volatile memory. They provide low standby power, high data density, fast access and excellent scalability. In the past decade, there has been significant research effort on engineering various types of memory device. Modeling of these memory devices including nominal performance and variability is highly demanded for design practice.

Phase Change Random Access Memory (PRAM) is one promising candidate for the next generation non-volatile memory. It has been shown to have excellent scalability, fast read access time, good data retention and high data density (Burr et al. 2010, 28:223-262, Wong et al. 2010, 98 :2201-2227). Figure 3.1 (a) shows the basic structure of PRAM. It consists of phase change material (chalcogenide alloy, $Ge_2Sb_2Te_5$ (GST)) and a heater. The phase change material can switch between the amorphous phase with high electrical resistance (logic ‘0’) and the crystalline phase with low electrical resistance (logic ‘1’). The phase is changed by increasing the temperature of phase change material with applied current pulse. To change the crystalline phase to amorphous phase, a current pulse with high amplitude and short duration is applied. To change the amorphous phase to crystalline phase, a lower and longer current pulse is applied. Figure 3.1 (b) shows the temperature profile in PRAM cell during these two switching periods. Additionally, by changing the current pulse profile, it is able to program PRAM to intermediate states continuously so that Multi-Level-Cell (MLC) is available for PRAM.



(a)



(b)

Figure 3.1. PRAM fundamental. (a) A basic PRAM cell structure. (b) Temperature profile of phase change material during programing.

Besides PRAM, Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) is another promising memory technology. It promises good combinations of high density, fast read and write access, low switching power, and non-volatile data storage (Wang et al. 2008, 44:2479-2482, Sharad et al. 2012, 11:843-853). STT-MTJ is based on Magnetic Tunnel Junction (MTJ). Figure 3.2 illustrates the basic structure of MTJ. Magnetic tunnel junction consists of two layers of ferromagnetic material separated by an insulator layer. The magnetic orientation of one layer is fixed while the magnetic orientation of the other layer can be changed by application of current pulse through MTJ. The magnetic angle between these two layers (parallel (P) or anti-parallel (AP)) determines the resistance of MTJ. When a current is applied through the junction, the spin of the electrons is polarized by the fixed magnetic layer, and then transferred to the free layer to change the magnetic orientation.

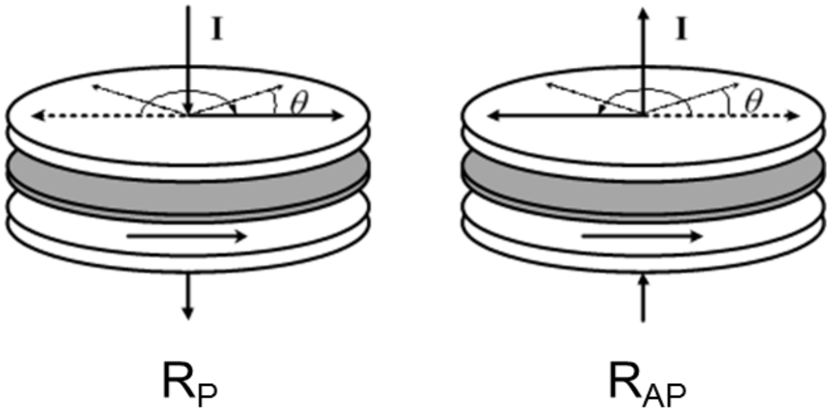


Figure 3.2. Magnetic Tunnel Junction structure.

A good memory model should have the following features. (1) The model should be based on physical principle of the device for it to be trustable and tunable with technology evolution. (2) The model should capture various performance metrics of memory device including static and dynamic behavior, energy, speed, variability, reliability and scalability to support optimizations at multiple design levels (system level, circuit level, etc.). (3) The model should be easily implemented in SPICE with high simulation efficiency for co-design with CMOS devices. Hence, a hierarchical memory modeling framework is proposed, which is capable with all the features above. Figure 3.3 illustrates the proposed hierarchical memory modeling framework, which is general for all types of memory devices. It starts from the common behavioral model of a digital memory, the finite-state-machine (FSM), with electrical forces to control the program/erase/hold of the state. The FSM is then mapped to a structural model, using an equivalent circuit for SPICE simulation. This includes device-level models that can capture the underlying physical mechanisms of phase change and the dependence on material/structure parameters. The physical nature of device-level model further helps embed variability and reliability issues in the analysis. From top down to the bottom, one can develop the model layer by layer, from abstraction to details. It also helps us with memory design from a system perspective. From bottom up, one can do optimization for the memory from a device perspective. With such hierarchical approach, cross-layer analysis and the comparison of different memories in each hierarchy level are enabled. This will give us a better understanding of different types of memory and make heterogeneous design easier. PRAM and STT-MRAM are then modeled within such a framework.

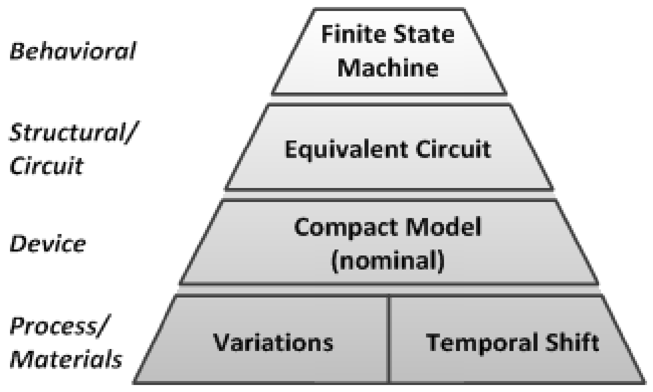
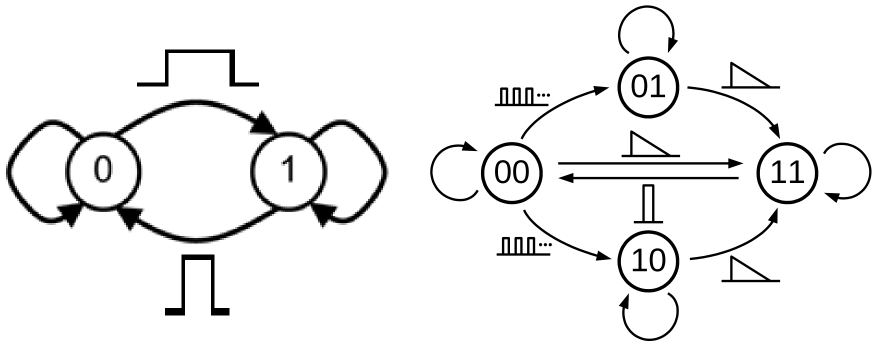


Figure 3.3. Hierarchical memory modeling framework.

3.2. Phase Change Random Access Memory

3.2.1. PRAM Modeling

At behavior level, PRAM is modeled as the finite-state-machine in Figure 3.4. Since PRAM has both single-level cell (SLC) and multi-level cell (MLC) application, it has multiple FSMs. As shown in Figure 3.4 (a), SLC PRAM has two digital states, '0' and '1'. To RESET the cell from '1' to '0', a high and short current pulse is applied. To SET the cell from '0' to '1', a lower and longer current pulse is applied. Since PRAM needs large write energy, the data is read before write; if the initial state and the target state are the same, no writing is needed. Figure 3.4 (b) shows the FSM model for a 4 levels MLC PRAM (Bedeschi et al. 2009, 44:217-227). Here '00' is high resistance amorphous state, '11' is low resistance crystalline state, and '01' and '10' are the two intermediate states. To SET PRAM to '11' state from any other initial state, a ramping down SET pulse is applied. To RESET a '01' or '10' state to '00' state, it needs to SET to '11' first and then RESET. To write '01' or '10', it needs to RESET to '00' first and then program in a read and verify process using sequential short current pulses. With the FSM, one can easily figure out the transfer of states under a given current pulse.



(a) (b)

Figure 3.4. The FSM models of PRAM. (The programming current waveform is illustrated.) (a) SLC (b) 4 levels MLC.

To map the FSM behavior into a circuit-level model for SPICE, an equivalent circuit model is introduced that captures each state transition in FSM. The input at this level is the initial state and the applied current pulse, and the output is the next state as well as the energy and latency of each programming step. The circuit level model focuses on the physical procedure of state transition, and is shared by both SLC and MLC operations.

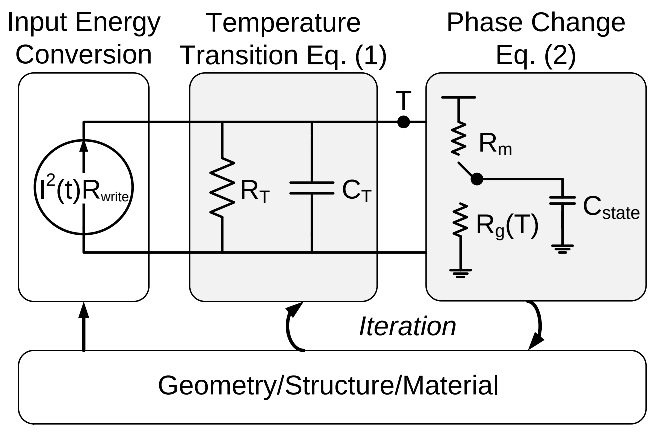


Figure 3.5. The equivalent circuit model of PRAM.

Table 1. Parameters in the PRAM circuit level model.

Parameters	Description
I	Input current amplitude
R_{write}	Electrical resistance of PRAM cell during write
R_T	Thermal resistance of GST
C_T	Thermal capacitance of GST
C_{state}	Store the state of memory cell
$R_g(T)$	Describe the crystallization of GST

Figure 3.5 gives the equivalent circuit model of PRAM, which consists of four parts, input energy conversion, temperature transition, phase change and geometry. Table 1 defines the parameters used at this level. The geometry block in Figure 3.5 describes the cross-sectional shape of a PRAM cell; the exact dimensions of each part are used to calculate electrical and thermal parameters. As the phase change of PRAM is based on heating, the input power is calculated by I^2R_{write} in the input energy conversion block. Since the crystalline GST has a very low resistivity, R_{write} is mainly the resistance of the metallic

heater (Itri et al. 2004). The energy conductance in PRAM is described by the energy conservation equation:

$$\int \left(I^2 R_{write} - \Delta T / R_T \right) dt = C_T \Delta T \tag{1}$$

where ΔT is the temperature difference between the top electrode and the interface of GST and the heater. Equation (1) is modeled by the $R_T C_T$ circuit in the temperature transition block (Figure 3.5), where R_T is the thermal resistance and C_T is the thermal capacitance (Kwong et al. 2008, Warren et al. 2008). The output node of this block indicates the temperature in the PRAM cell, which is further used in the phase change block. The phase change block consists of a capacitor C_{state} to indicate the state of the memory, a switch, a voltage source and resistors. The temperature evaluated by the temperature transition block is used to decide the switch position: when the temperature is higher than melting temperature ($T>T_m$), the switch flips up and C_{state} is charged by the voltage source, indicating the melting of GST. When the temperature is between the melting and annealing temperature ($T_a<T<T_m$), the switch flips down and C_{state} is discharged through R_g, indicating the annealing of GST. The crystallization process is described by the Johnson-Mehl-Avrami (JMA) equation (Johnson and Mehl 1939, 135:416-458):

$$f = 1 - \exp\left(- t \big/ \tau(T) \right) \tag{2}$$

where f is the fraction of the crystalized region, t is time, and τ(T) is a temperature dependent time constant defined by:

$$\tau(T) = \begin{cases} R_m C_{state} & (T > T_m) \\ R_g(T) C_{state} = \tau_0 \exp\left(E_A \big/ k_B T \right) & (T_a < T < T_m) \end{cases} \tag{3}$$

where E_A is the active energy of crystallization, k_B is Boltzmann constant and τ_0 is a constant. Therefore, the RC circuit models the exponentially phase change of PRAM and the voltage of C_{state} indicates the size of amorphous region, which corresponds to the memory state.

This abstracted level of PRAM model can be used to capture the nominal performance of single memory cell. The latency of write is defined as the time when the input current is applied to the end when the cell cools down. So, the latency can be obtained from the temperature transient curve by the circuit model. The energy of write is evaluated by $I^2R_{write}t_{pulse}$, where t_{pulse} is the time period of the input current pulse.

The device level model of PRAM predicts the parameters in the circuit model from cell geometry. It further paves a path to analyze the variability issues. TCAD simulators can be used to analyze the resistance change of PRAM, but with poor simulation efficiency. Thus, a geometry based compact model is necessary for design analysis. In this section, a geometry dependent resistance model is proposed for the mushroom cell PRAM (Wong 2010 98:2201-2227), which is one of the most common shapes of PRAM cell. Raphael 2D, a TCAD tool, is used to validate the model.

Figure 3.6 shows the 2D structure and the potential profile of a mushroom cell PRAM from Raphael 2D. The key dimensions of a mushroom cell are also specified in Figure 3.6: d is the thickness of GST, W is the width of the top contact, and CW is the width of the bottom contact between GST and the heater. It is assumed that the shape of the amorphous region is a hemisphere with radius r. These dimensions will determine the electrical resistance, thermal resistance and thermal capacitance of PRAM memory cell.

Table 2 summarizes the material properties that are used in this model (Li and Chan 2008, Li et al. 2008, 7:138-141).

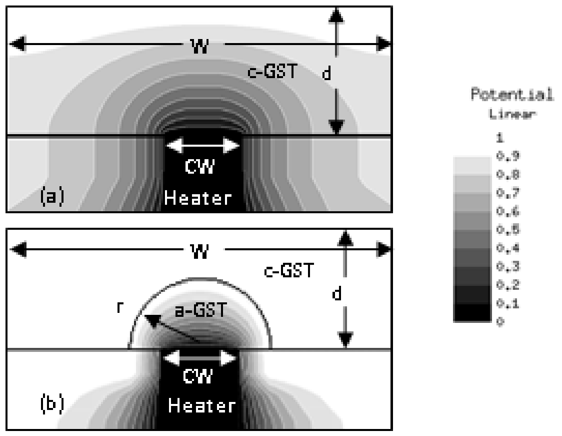


Figure 3.6. 2D structure of PRAM cells in Raphael. (a) Fully crystalline state. (b) Fully amorphous state.

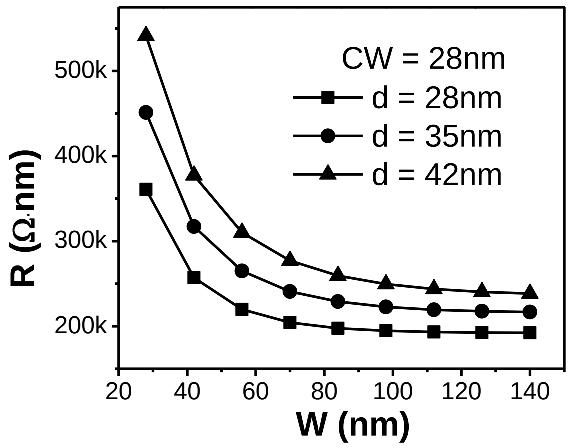
Table 2. Material properties in PRAM model (Li and Chan 2008, Li et al. 2008, 7:138-141).

Material	Electrical resistivity ρ (Ω·cm)	Thermal conductivity κ ($W \cdot m^{-1} \cdot K^{-1}$)	Specific Heat c_0 ($J \cdot cm^{-3} \cdot K^{-1}$)
Crystalline GST	0.0361	0.5	1.25
Amorphous GST	33.33	0.2	1.25

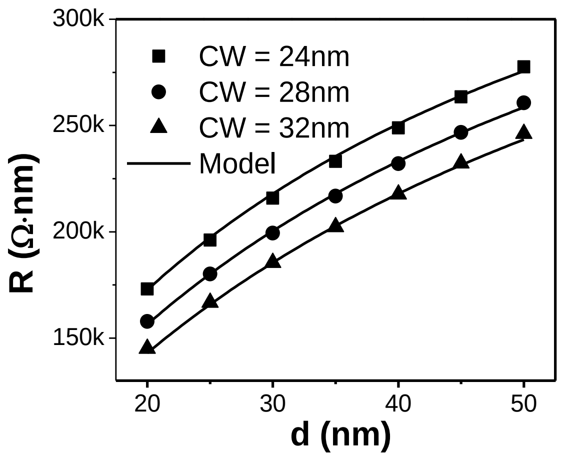
Figure 3.7 shows the dependence of the resistance of fully crystalline GST on W, CW and d. Figure 3.7 (a) shows the saturation of resistance with increasing W. Thus, our model only focuses on the case when W is large enough and does not affect the resistance. Since the two contacts of GST are of different size, we use an effective width to calculate the resistance as:

$$R = \rho_c \frac{d}{\alpha CW} \tag{4}$$

where the coefficient α is the effective factor calculated as α = 0.79*d/CW+1.08. Figure 3.7 (b) validates the model for the crystalline resistance.



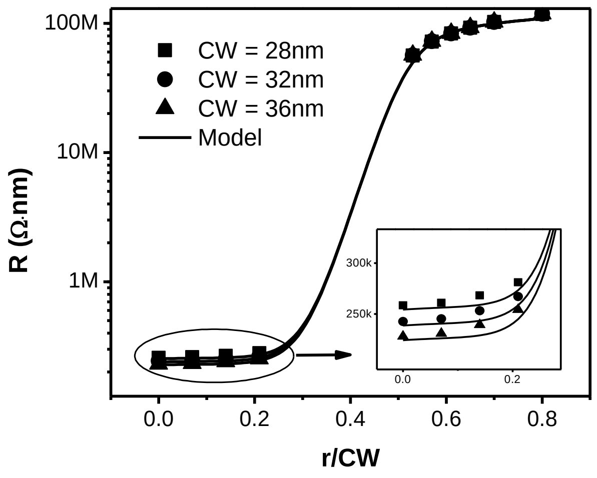
(a)



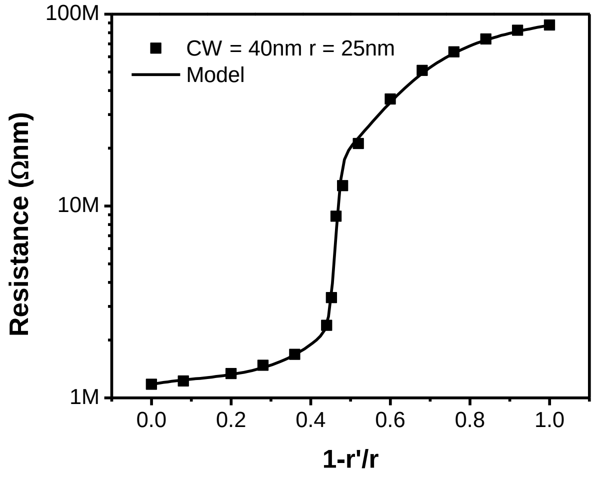
(b)

Figure 3.7. Geometry dependence of fully crystalline GST resistance. (a) Saturation of R with W increasing. (b) Fully crystalline resistance model vs. Raphael simulation results.

For the amorphous state, as shown in Figure 3.8 (a), the resistance primarily depends on r/CW, due to the large difference in the resistivity of amorphous GST and crystalline GST. When the mushroom fully covers the bottom contact (r/CW > 0.5), GST shows a much higher resistance than that in the crystalline state. When the mushroom is very small, the resistance is low and does not change much with r/CW. The resistance changes dramatically when r/CW ≈ 0.5. We use a smoothing function to continuously model the resistance.



(a)



(b)

Figure 3.8. Geometry dependent electrical resistance model. (a) RESET (b) SET

When the phase change material is changing from amorphous state to crystalline phase (SET), a smaller crystalline mushroom (with a radius of r') grows up inside the original amorphous mushroom from the bottom contact. Figure 3.8 (b) shows the resistance change during this process. The difference in resistance value is due to the different resistivity for the two figures.

The thermal resistance of GST is modeled using a similar approach. The thermal resistance of full crystalline GST is calculated by:

$$R_T = \frac{d}{\kappa_c \alpha CW} \tag{5}$$

On the other side, since the difference in thermal conductivity of crystalline GST and amorphous GST is not as much as that of electrical resistivity, thermal resistance R_T does not change much with r/CW, as shown in Figure 3.9 (a). The thermal resistance also depends on d/CW in both crystalline state and amorphous state. Thermal capacitance C_T depends on d and CW because crystalline GST and amorphous GST have the same heat capacity, as shown in Figure 3.9 (b):

$$C_T = c_0 \alpha CWd / 2 \tag{6}$$

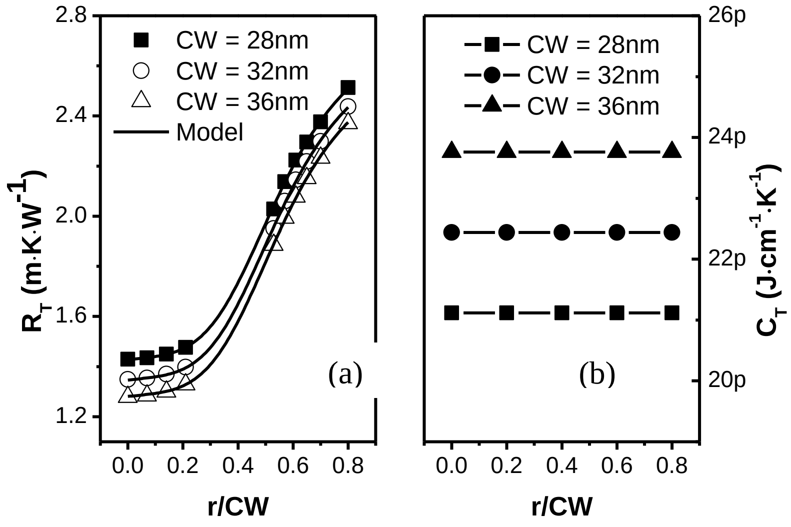
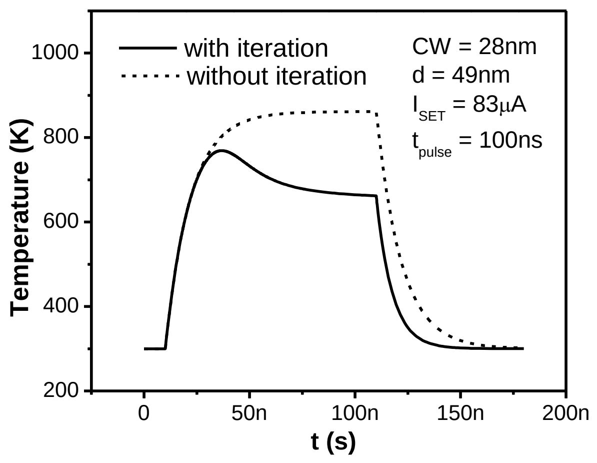
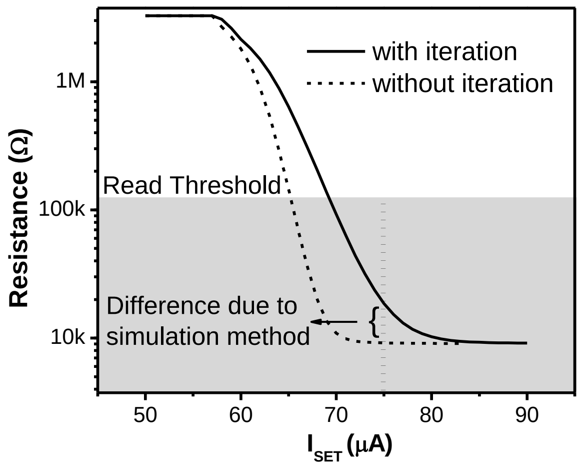


Figure 3.9. Compact models of thermal parameters.



(a)



(b)

Figure 3.10. The iteration in model simulation significantly impacts the prediction of (a) temperature profile and (b) resistance transition.

With these geometry dependent resistance and capacitance models, the parameters in the circuit level model (Figure 3.5) can be obtained. Since R_T depends on the state of PRAM, which is represented by the size of the amorphous region, it changes during the SET process. Thus, it is important to iteratively simulate the heat and phase transition blocks (Figure 3.5) in order to accurately determine the temperature profile and the final resistance value, as shown in Figure 3.10.

The resistance of a PRAM cell is predicted from the model with given initial geometry information and input current pulse, the geometry, and the material property. This model matches with experimental data from the paper of Lacaita et al. (2004) as shown in Figure 3.11. CW=68nm and d=119nm is used in this simulation.

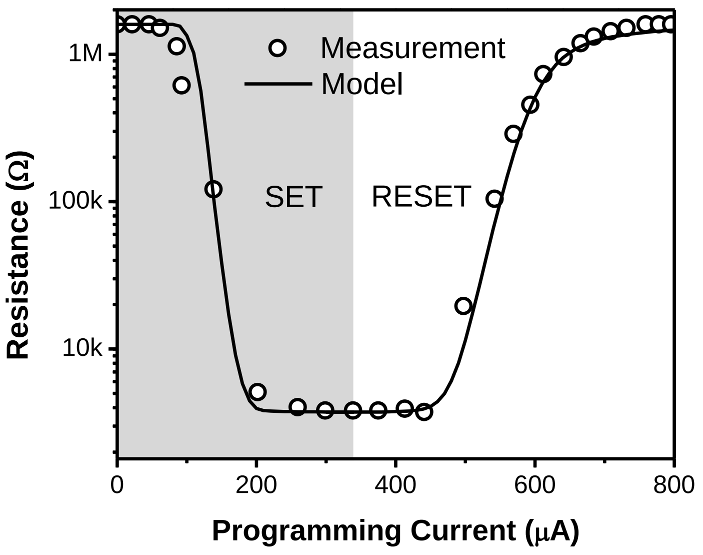


Figure 3.11. R-I characteristics validated with experimental data from (Lacaita et al. 2004). CW=68nm and d=119nm.

In the next level, process variation and material reliability are modeled. Since our device level model is geometry dependent, the impact of process variation can be easily simulated, which will be further discussed in Section III. There are two main reliability issues for PRAM, soft errors due to resistance drift and Stuck-SET failures. While error due to resistance drift can be recovered by the write process, Stuck-SET is hard error that is attributed to Ge depletion (Kim and Ahn 2005) or Ge contamination at the interface of GST and the heater. The resistance drop is modeled as

$$\Delta R = aN^b \tag{7}$$

where N is the number of programming cycles, a and b are fitting parameters as shown in Figure 3.12. In the circuit level, it is modeled by the degradation of the heating efficiency. In order to emulate this effect, the input energy I^2R_{write} is multiplied by a coefficient λ, which is less than 1 and is a function of N.

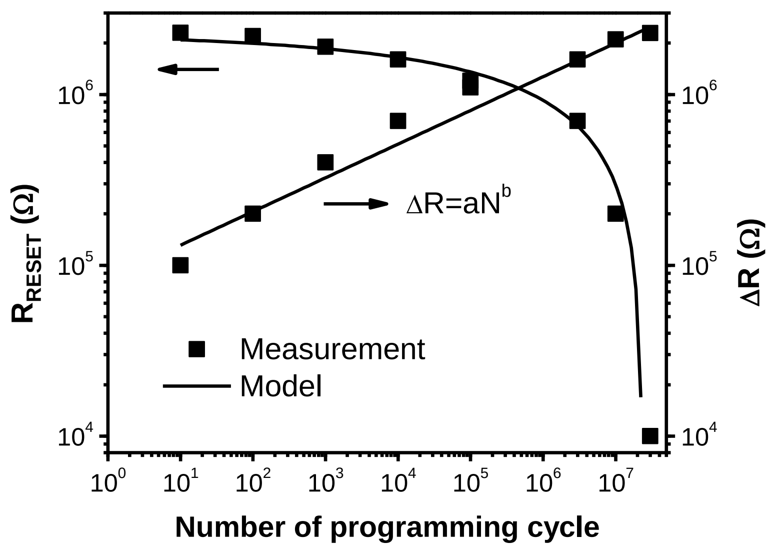


Figure 3.12. Stuck-SET model fitted with measured data from (Kim and Ahn 2005).

3.2.2 PRAM Design Exploration

In this section, the performance of the proposed model is analyzed with respect to variability and reliability for the 32nm technology node. A new metric, State Transition Curve, is presented to capture the characteristics of PRAM for reliable design. PRAM employs 1T1R (1 transistor 1 resistor) structure as shown in Figure 3.13. BL, WL and SL correspond to bit line, word line and source line respectively. The models are incorporated into HSPICE using Verilog-A to simulate the shown structure. PTM 32nm HP (High Performance) MOSFET (Zhao and Yu, 2006, 53:2816-2823) is used for the simulation in order to get high current for programming. Critical parameters of interest including their variations are listed in Table 3.

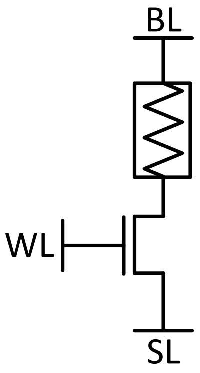


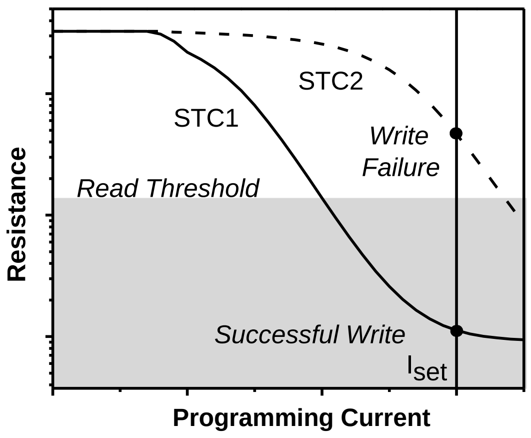
Figure 3.13. PRAM cell structure

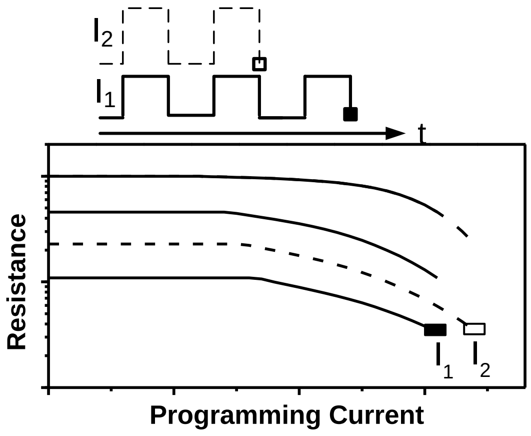
Table 3. PRAM cell simulation parameters

	Parameter	Value (μ±3σ)
PRAM	CW	28 nm ±4%
	d	49 nm ±2%
	R_{SET}	9 kΩ
	R_{RESET}	3.6 MΩ
	R_{write}	1 kΩ
CMOS	V_{dd}	0.9 V
	V_{th}	494mV ±45mV
	Length	28 nm

For PRAM, State Transition Curve (STC) is a curve that describes resistance changes with programming current amplitude for a particular pulse width. Since the currents for SET and RESET are quite different, there are two sets of STC for PRAM. Figure 3.14 (a) shows the STC for SET and indicates the role of STC in reliable PRAM design. STC1 and STC2 show the characteristics of PRAM under nominal and variational condition respectively. The initial resistance is determined by the initial geometry of PRAM. The resistance transition is determined by the phase change model in Figure 3.5

and the geometry dependent resistance model in Figure 3.8. If the final resistance is a lot larger than the read threshold resistance, we can choose the current that achieves a successful write. However, such a choice may not be sufficient due to variability. STC2 in Figure 3.14 (a) represents incomplete state transition due to variation. If we still choose the same current, the write fails. But in MLC programming, intentional incomplete state transition is adopted by using sequential short current pulses as shown in Figure 3.14 (b). We can tune the current pulse to obtain the required resistance value. Figure 3.14 (b) shows that different magnitude of input current can achieve the same resistance with different number of current pulses. It consumes more energy for more programming steps (I_1), but smaller resistance steps help more accurate MLC resistance control.

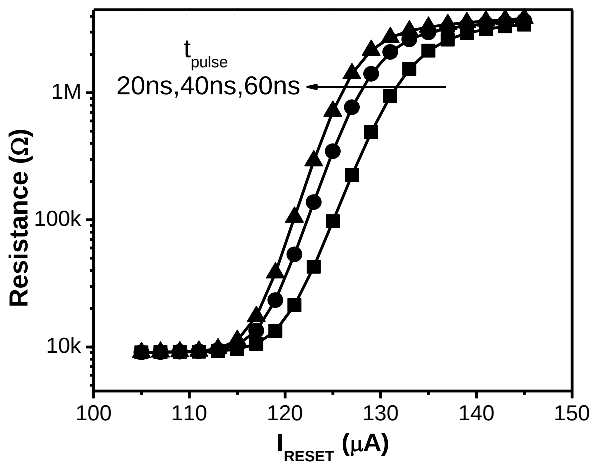




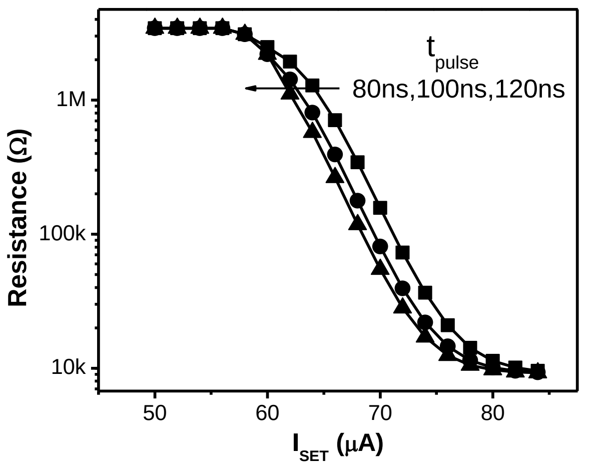
(a) (b)

Figure 3.14. State Transition Curve of (a) SLC SET and (b) MLC SET.

In this section, the performance of SLC PRAM cell is analyzed with proposed model and STC. We extract results under CMOS and GST process variation and temporal degradation. For nominal condition, STCs for different pulse width are presented in Figure 3.15. When t_{pulse} increases, STCs shift left, so that smaller current is needed to get the same resistance. Thus, we can choose current and pulse width as listed in Table 4. The energy and latency performance of PRAM are also presented. The latency is obtained by the temperature transient curve, which is larger than the pulse width.



(a)



(b)

Figure 3.15. State Transition Curve in the nominal condition with different pulse width: (a) SET (b) RESET.

Table 4. Nominal performance of 32nm SLC PRAM.

	SET	**RESET**
I (μA)	85	140
t_{pulse} (ns)	100	40
R (Ω)	9.3k	3.6M
Energy (pJ)	0.72	0.78
Latency (ns)	107	50

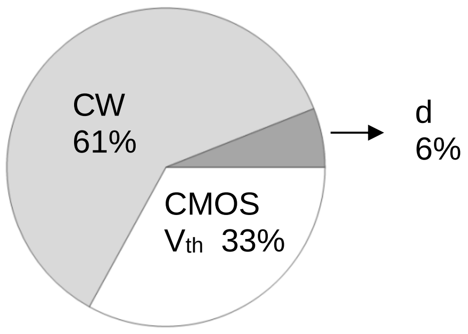


Figure 3.16. The contribution of different variation sources to the variability of R_{RESET}.

Next, the effect of variation is considered. Three variation sources are considered in this work: V_{th} of the transistor, CW and d as listed in Table 3. From the simulation, the variations of PRAM resistance (ΔR_{SET} and ΔR_{RESET}) are obtained. ΔR_{RESET} is much larger than ΔR_{SET}. The contribution of the variation sources to ΔR_{RESET} is shown in Figure 3.16. We see that CW is the critical variation source. Further, we analyze the impact of variation in CW on memory performance. Figure 3.17 shows how STC changes with CW. As CW increases, STC shifts right and R_{RESET} drops at a fixed I_{RESET}. Under nominal CW, when I_{RESET} = 130μA, R_{RESET} is much higher than read threshold indicating a successful write. For the same current, when CW experiences variation of +4%, R_{RESET} is below read threshold causing a write failure. However, if I_{RESET} = 140μA, we always have a successful write under variation of CW. Therefore, this plot helps choose RESET current with a fixed pulse width to tolerate different CW variations.

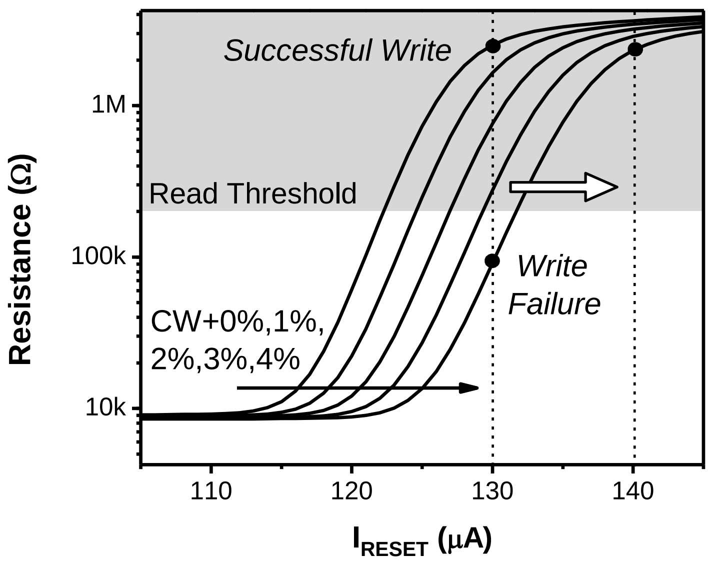


Figure 3.17. RESET State Transition Curve is highly sensitive to CW variations ($t_{pulse} = 40ns$).

Figure 3.17 also shows that if we fix the margin from R_{RESET} to the read threshold resistance, we have to increase I_{RESET} with CW. For fixed resistance margin, the write power, I^2R_{write}, increases with CW as shown in Figure 3.18. In addition, the power goes up when we reduce the current pulse width since the required current is larger. Figure 3.19 shows the relation between required RESET current and pulse width for maintaining different resistance margins for successful writes. In order to achieve higher resistance margin, more current and time are needed resulting in higher power requirements.

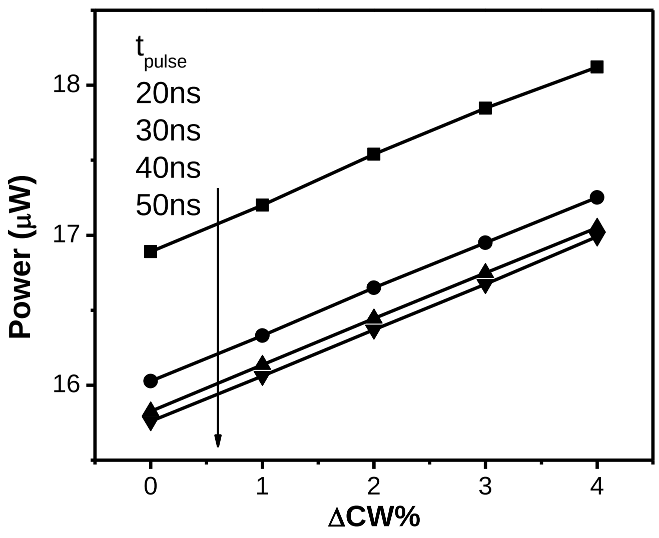


Figure 3.18. Write power dramatically increases with larger CW variations, assuming a constant resistance value after write.

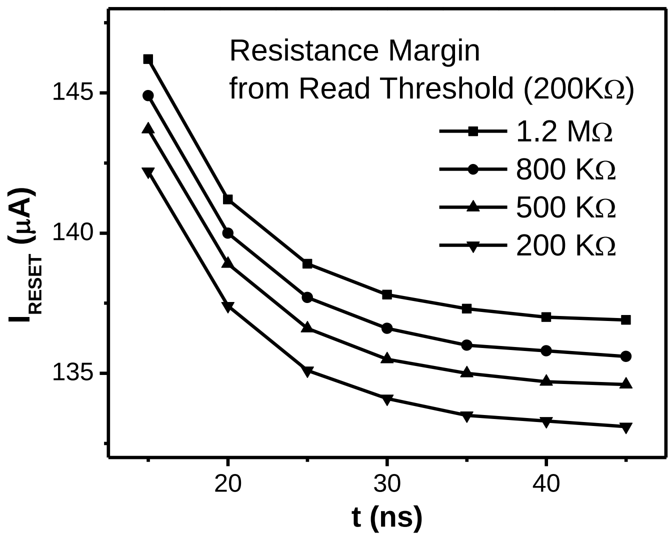


Figure 3.19. Required RESET current and pulse width with given resistance margin.

Finally, the impact of Stuck-SET on STC is simulated and the results are presented in Figure 3.20. The degradation increases as the number of cycles increases. If I_{RESET} =

133μA, RESET resistance of PRAM drops below the read threshold after 10^6 cycles, which causes hard error. We can increase I_{RESET} to 140μA to increase PRAM lifetime to 10^7 cycles.

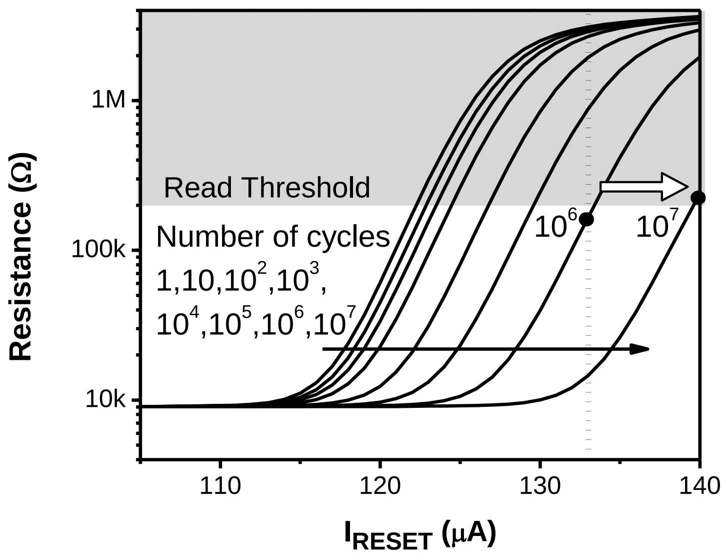


Figure 3.20. State Transition Curve with Stuck-SET degradation.

3.3. Spin-Transfer Torque Magnetic Random Access Memory

3.3.1. STT-MRAM SPICE Model

Since STT-MTJ has only two stable states, STT-MRAM can only support SLC application. The FSM is the same as SLC PRAM as shown in Figure 3.4 (a).

The dynamics of the magnetic moment $\vec{M}$ of the free layer in MTJ is defined by the LLG equation (Ralph and Stiles 2008, 320:1190-1216, Kammerer et al. 2010, 57:1408-1415):

$$\frac{d\vec{M}}{dt} = -\gamma\mu_0\vec{M}\times\vec{H}_{eff} - \gamma\frac{2K}{M_s^2}\left(\vec{M}\cdot\vec{u}_{ea}\right)\cdot\left(\vec{M}\times\vec{u}_{ea}\right) + \frac{\alpha}{M_s}\vec{M}\times\frac{d\vec{M}}{dt} + \eta\frac{\mu_B I}{eV} \tag{8}$$

The terms (from left to right) represent the Zeeman torque, by both the local field and the thermal fluctuation field, the anisotropic torque, the damping torque, and the spin-transfer torque, in which the efficiency (η) depends on the current direction. Table 5 defines key model parameters (Lu et al 2007, 40:320-325, Faber et al. 2009).

Table 5. Parameters on the geometry and materials.

Saturation magnetization (M_s)	$M_{s0}\left[4\left(1-\frac{1}{4r/ch-1}\right)\cdot\exp\left(-\frac{2S_b}{3R}\frac{1}{4r/ch-1}\right)-3\right]$	
Effective magnetic field ($\vec{H}_{eff}$)	Intrinsic $\vec{H}$ +thermal fluctuation $\vec{H}_f$	
Gyromagnetic ratio (γ)	Anisotropic const. (K)	Damping const. (α)
$1.76\text{x}10^{11}$ rad·s^{-1}/T	$1.13\text{x}10^{5}$ T·A/m	0.02

The switching of the magnetic moment is the key dynamics in STT-MTJ. In general, it is a three dimensional movement: the Zeeman torque and the anisotropic torque contribute to the rotation in the plane perpendicular to the easy axis, indicated by an angle φ; the damping torque and the spin-transfer torque dominate the switching in the easy plane, resulting in the change of θ. Considering a realistic structure of STT-MTJ (Figure 3.21), the change of the magnetic moment can be separated into two planes and thus, the LLG equation reduced to two scalar equations of magnetic angle φ and θ.

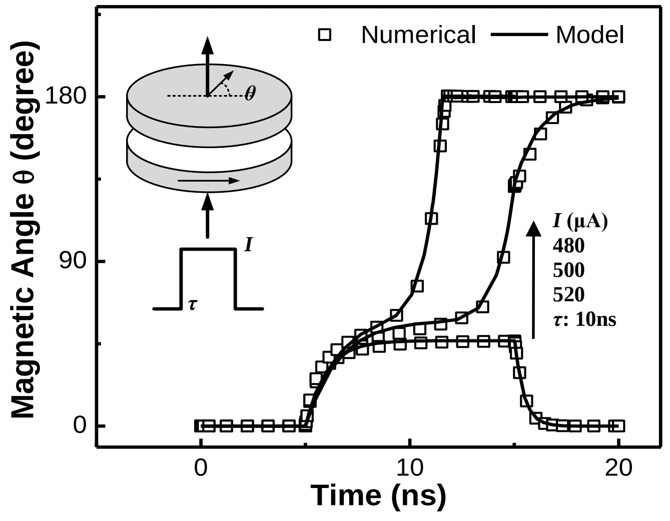


Figure 3.21. An in-plane STT-MTJ is programmed by a current pulse.

$$M_s \frac{d\varphi}{dt} = -\gamma\mu_0 M_s H \sin\theta - 2\gamma K \sin\theta\cos\theta \tag{9}$$

$$M_s \frac{d\theta}{dt} = \alpha M_s \frac{d\varphi}{dt} + \eta \frac{\mu_B I}{eV} \tag{10}$$

Substituting *dφ*/*dt* from Equation (9) to Equation (10):

$$M_s \frac{d\theta}{dt} = -\alpha\gamma\left(\mu_0 M_s H \sin\theta + 2K \sin\theta\cos\theta\right) + \eta \frac{\mu_B I}{eV} \tag{11}$$

The scalar equation of Equation (11) is the foundation to analyze the switching dynamics of *θ*. Based on Equation (11), Figure 3.22 plots *dθ*/*dt* for different *I* when *θ* changes from 0° to 180°. Some critical points are highlighted below in order to obtain the physical map for further model derivation:

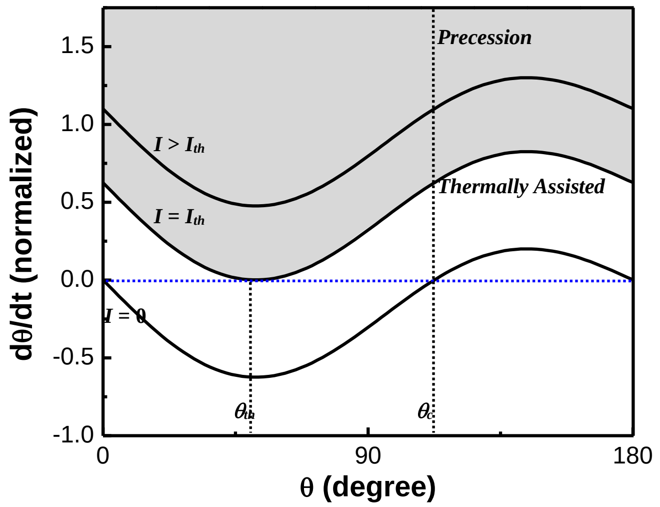


Figure 3.22. *dθ*/*dt* for different *I* when *θ* changes from 0° to 180°.

Threshold current (I_{th}): This concept separates two possible switching mechanisms in a STT-MTJ device, precession switching and thermally assisted switching. When $I > I_{th}$, *dθ*/*dt* is always > 0 (Figure 3.22) and thus, the magnetic angle is able to complete the switching with sufficient *τ*. However, when $I < I_{th}$, *dθ*/*dt* may be < 0, requiring the assist of thermal fluctuation H_f to statistically switch STT-MTJ (Faber et al. 2009). As the thermal process takes a longer time and is not deterministic, today's STT-MTJ usually follows the precession switching. I_{th} can be solved from the minimum of $d\theta/dt = 0$, which is associated with the threshold angle θ_{th} (Figure 3.22).

Critical angle (θ_c): This angle defines a critical value the magnetic moment has to reach at the end of the current pulse; if θ at time = τ is smaller than θ_c, the damping torque may pull θ_c back to 0^o (Figure 3.21). As observed in Figure 3.22, when $I = 0$, there are three points to satisfy $d\theta/dt = 0$: 0^o and 180^o are two stable solutions, while θ_c is a metastable point. This behavior is similar as that in a SRAM cell, and helps us develop the model of θ_c.

Critical current (I_c): Given the pulse width τ, I_c is the minimum current required to switch the magnetic angle from 0^o to θ_c. $I > I_c$ ensures a successful precession switching. To solve I_c, Equation (11) is integrated from 0^o to θ_c for dθ, and from 0 to τ for dt. A compact solution is obtained (Table 6). I_c is proportional to the inverse of τ, implying a tradeoff between speed and the writing power in design optimization.

Table 6 summarizes the models for $P \rightarrow AP$, in standard international units. For $AP \rightarrow P$, the formulas remain the same, but with different coefficient values due to the different initial condition. The formulas in Table 6 have a clear root in physics, and are accurate in the precession switching. They are scalable with process and material parameters, supporting the development of the RC network. Figure 3.23 validates our model with the measurement data (Chun et al. 2013, 48:598-610). Due to the operation nature of STT-MTJ, it requires more current and energy to switch it from P to AP (Figure 3.23).

Table 6. Models of Critical Points in STT-MRAM.

θ_{th}	$\cos^{-1}\left[\left(\sqrt{(\mu_0 M_s H)^2 + 32K^2} - \mu_0 M_s H\right)/(8K)\right]$
I_{th}	$\alpha\gamma(eV)/\eta\mu_B \cdot [\mu_0 M_s H \sin\theta_{th} + K\sin(2\theta_{th})]$
θ_c	$\cos^{-1}[-\mu_0 M_s H/(2K)]$
I_c	$(\pi r^2)\cdot(2.1\times10^{-4} M_s/\tau + 0.34 M_s H + 4.26\times10^{10})$

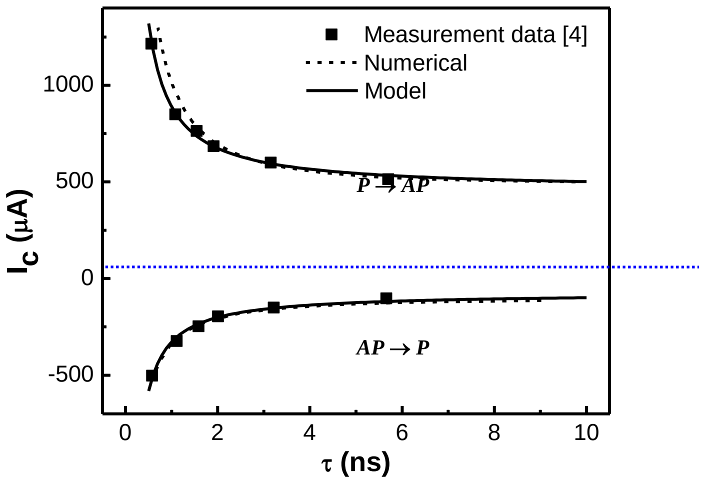


Figure 3.23 Validation with published STT-MTJ data (Chun et al. 2013, 48:598-610). (r = 45nm, T_{ox} = 0.85nm)

The switching of the magnetic angle represents the write process of STT-MTJ. The read of the state is by characterizing the resistance of MTJ. When a read current is delivered to STT-MTJ, the resistance reaches a low value (R_P) if the magnetic moments in both ferromagnetic layers are in parallel; otherwise a higher resistance (R_{AP}) is detected. Coupled with the dynamic magnetization procedure in previous section, this property completes the operation of STT-MTJ.

The tunnel magneto-resistance (*TMR*) of MTJ is defined as (R_{AP}-R_P)/R_P. During the continuous switching of the magnetic angle, the change of MTJ resistance follows (Madec et al. 2010, 57:1416-1424):

$$R(\theta) = 2R_P\left(\frac{1+TMR}{2+TMR+TMR\cdot\cos\theta}\right) \tag{12}$$

The static values of R_{AP} and R_P are calculated from the tunneling current through T_{ox}. Equation (12) is used to model the dynamic resistance during the switching period.

3.3.2. STT-MRAM Compact Model

The details of the switching period are important for various design purpose, such as power and yield. In addition, design applications of STT-MTJ usually involve CMOS as the control device. For these reasons, compact model of STT-MTJ needs to be embedded into the SPICE simulator. Different from previous approach that directly implement the LLG equation through complex Verilog-A codes, this work also proposes a simple RC network that is physical, intuitive, and general.

Starting from the fundamental LLG equation (Equation (11)), sinθ can be approximated as θ, 1, or -θ, when θ is close 0^o, 90^o, or 180^o, respectively. A similar treatment can be applied to cosθ. By expanding sinθ and cosθ in this approach, $d\theta/dt$ in Equation (11) is expressed as a linear function of θ, and thus, the solution of the LLG equation is transferred as a passive RC network for SPICE simulation.

Based on this general principle, four distinct regions are recognized, easing the implementation. Figure 3.24 shows the network which supports transient SPICE simulations, with the output node representing θ. *R*s are functions of those critical points in Table 6, and *C* is a constant, as derived below:

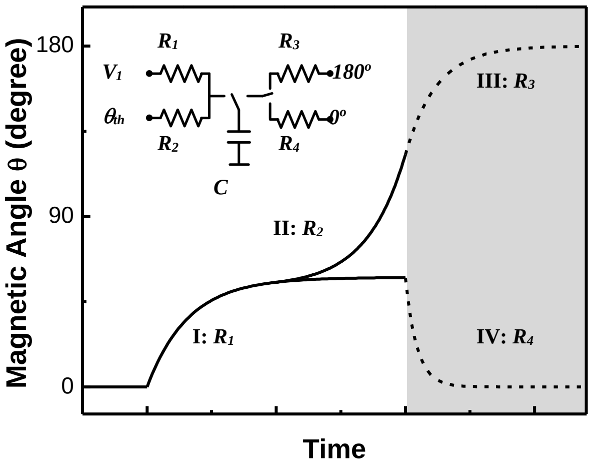


Figure 3.24. The regional RC network for dynamic SPICE simulation.

Region 1: This is at the beginning of the current pulse, when θ is close to 0 and thus, $\sin\theta \sim \theta$ and $\cos\theta \sim 1$. The damping torque resists the change of θ, implying that the R_1C network is a negative feedback. V_1, which is a linear function of applied I, is charging the output node.

Region 2: As soon as θ exceeds the threshold angle θ_{th}, θ is close to 90^o so that $\sin\theta \sim 1$ and $\cos\theta \sim 90^o - \theta$. In this region, $d^2\theta/dt^2$ becomes positive, as indicated in Figure 3.22. Such a fact suggests that the RC network is a positive feedback: the increase in θ helps speed up the switching. Therefore, a negative resistance, R_2, is obtained from Equation (11), giving an exponential increase in the magnetic angle (Figure 3.24). If the current pulse stays long enough, the magnetic angle rapidly reaches 180^o, as shown in Figure 3.21. However, if τ is not long enough to complete the switching, two more regions are needed for time > τ.

Region 3: If $\theta > \theta_c$ when the current pulse ends, the damping torque helps finish the switching without I, as shown in Figs. 3.21. In this case, Equation (11) can be expanded around 180^o to obtain R_3C.

Region 4: Finally, if $\theta < \theta_c$ when the current pulse ends, the damping torque overwhelms and pulls the magnetic angle back to the initial state, 0^o. The switching fails, under the influence of R_4C.

Table 7 summarizes all model parameters. They are in closed-form, derived from the LLG equation and parameters in Table 6. The proposed RC network is followed by the *TMR* model (Equation (12), in Verilog-A) to complete the simulation structure. Working together, they convert the magnetic angle to electrical resistance. As all parameter values are pre-solved before the simulation, this RC network is highly efficient in the SPICE environment.

Table 7. Formulas for the RC Elements in SPICE.

V_1	$\theta_{th} + a(I - I_{th})/(\pi r^2)$
R_1	$K_1(\mu_0 M_s H + 2K)/M_s$
R_2	$K_2[1 - b(I - I_{th})/(\pi r^2)]K/M_s$
R_3	$K_3(2K - \mu_0 M_s H)$
R_4	$K_4(\mu_0 M_s H + 2K)/M_s$

The newly developed models are implemented into SPICE. Two simulation examples are presented in Figure 3.21 and Figure 3.23. Under the same assumptions of *I*, *r* and T_{ox}, Figure 3.25 further demonstrates the prediction under different pulse width τ. As expected by the RC network in Figure 3.24, different RC components are activated, depending on the switching condition. The success of data writing is determined by both the magnitude and the duration of the current pulse. The proposed modeling and simulation method smoothly captures such a behavior for design exploration.

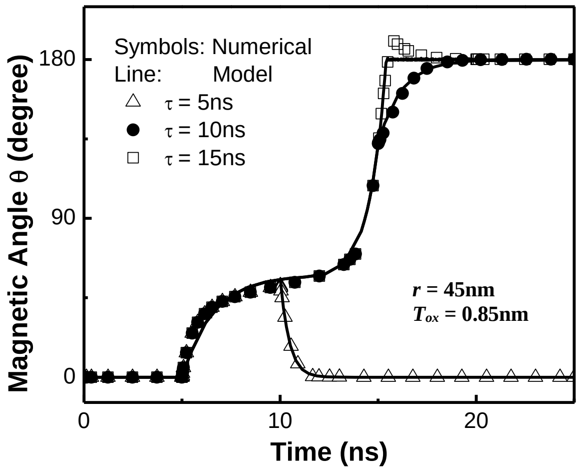


Figure 3.25. The switching behavior under different pulse widths.

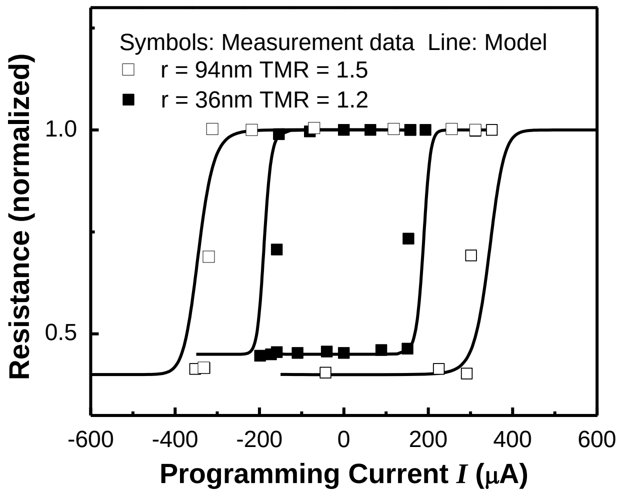


Figure 3.26. The matching in the prediction of MTJ resistance.

By combining the switching model and the *TMR* model together, the new solution generates the electrical property of STT-MTJ. Figure 3.26 validates this approach with the experimental data (Diao et al. 2007, 19:165209, Lin et al. 2009). For a STT-MTJ device, since $P \rightarrow AP$ starts from $\theta = 0^o$ but $AP \rightarrow P$ starts from $\theta = 180^o$, these two switching paths experience different switching thresholds, as predicted by the LLG equation. This causes the hysteresis behavior in the resistance, which is well matched by our proposed models. In addition, the new compact model is general enough to describe the data from different processes, as demonstrated in Figure 3.26.

3.3.3. STT-MRAM Design Exploration

The STT-MRAM models are implemented into SPICE with Verilog-A. The effects of technology scaling and the design under reliability constraints are studied.

From the perspective of technology scaling, Figure 3.27 examines the minimum programming current, Ic, under shrinking of device feature size for fixed pulse width τ = 5ns. The radius r impacts the density of Ic mainly through saturation magnetization (Ms in Table 5), which is a material property (Lu et al. 2007, 40:320-325). The density of Ic is sensitive to r only when the radius is smaller than 20 nm. On the contrary, Ic is highly sensitive to Tox, as Tox affects the intrinsic magnetic field in Equation (8). In addition, Tox has a strong influence on the resistance and the long-term reliability of the tunnel junction (Madec et al. 2010, 57:1416-1424). Therefore, process control of Tox is extremely important to STT-MTJ based memory design.

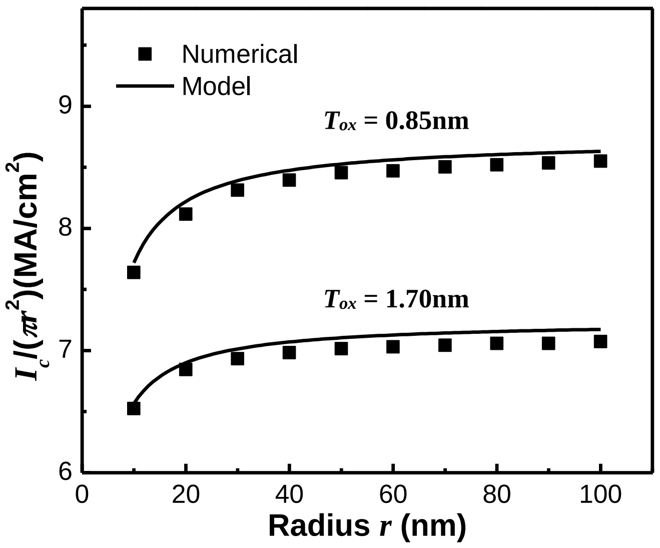


Figure 3.27. The scaling of critical current of STT-MTJ device.

Then, we investigate the design optimization of a single cell STT-MRAM under reliability constraints. Similar to PRAM, STT-MRAM adopts 1T1MTJ structure, as shown in Figure 3.28. BL, WL and SL correspond to bit line, word line and source line respectively. We used the MTJ model parameters in Figure 3.23 and PTM HP transistor model in 45nm technology (Zhao and Yu, 2006, 53:2816-2823). Critical parameters of interest including their variations are listed in Table 8.

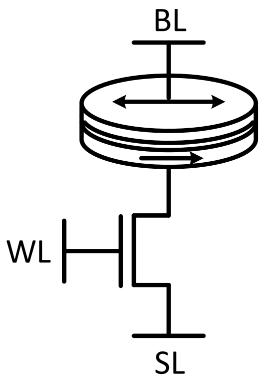


Figure 3.28. 1T1MTJ memory cell structure.

Table 8. STT-MRAM cell simulation parameters.

	Parameter	Value (μ±3σ)
STT	r	45 nm ± 1 nm
	T_{ox}	0.85 nm
	R_p	1 kΩ
	R_{ap}	2 kΩ
CMOS	V_{dd}	1 V, 2.2 V
	V_{th}	469 mV ± 5 mV
	Length	45 nm

Due to the difference in required critical current for P $\rightarrow$ AP and AP $\rightarrow$ P switchings, a boosted voltage of 2.2V is applied for P $\rightarrow$ AP while nominal Vdd of 1V is used for AP $\rightarrow$ P. In the memory cell, process variation affects both MTJ and access transistor. We use the MTJ radius r and transistor threshold voltage Vth to represent the variation sources listed in Table 8. After embedding these variations into the nominal model, one million Monte Carlo simulations are run to show the yield under different programming conditions.

We calculated the bit error rate (BER) as the percentage of the simulations in which the angle θ reaches 180o from 0o and vice versa at the end of simulation time. To guarantee reliability constraint of block failure rate (BFR) of 10^{-8}, all the current profiles described by the combination of current pulse amplitude (I) and width (τ), result in the same BER of $2x10^{-5}$. We set this BER constraint since BFR = 10^{-8} of a 512-bit block can be achieved with ECC scheme BCH (t=2) that results in small hardware overhead (Yang et al. 2014, 76:133-147).

Programming current and cell energy consumption are generated for P $\rightarrow$ AP and AP $\rightarrow$ P with equal BER in Figure 3.29 and Figure 3.30, respectively. The energy is calculated by the integration of I*Vdd for the duration of the pulse. It includes both STT-MTJ and transistor energy. We see for both P $\rightarrow$ AP and AP $\rightarrow$ P switchings, current pulse amplitude increases as current pulse width decreases, similar to the trend shown in Figure 3.23. However, under the BER = $2x10^{-5}$ constraint, the required current amplitude is larger than the critical current amplitude. Correspondingly, required transistor size has to be increased to support the large current amplitude. On the other hand, programming energy increases as current pulse width increases because the decrease in current I is slower than the increase in τ.

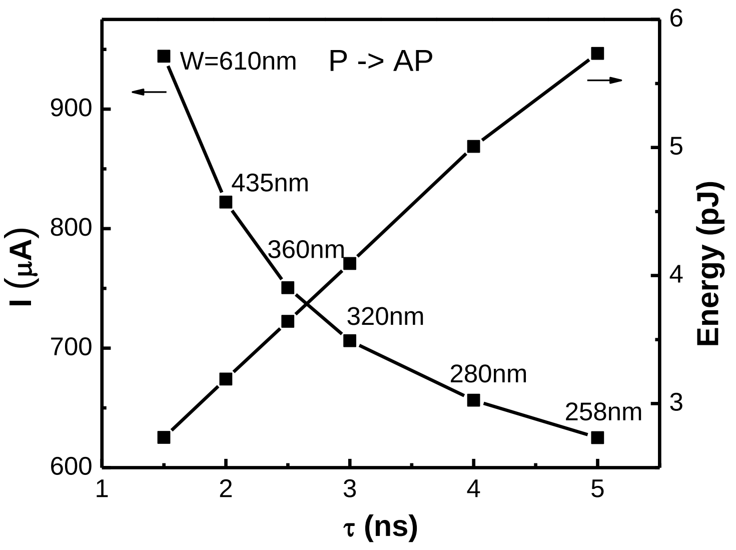


Figure 3.29. Programming current and energy of $P \rightarrow AP$ switching. All points have $2x10^{-5}$ BER.

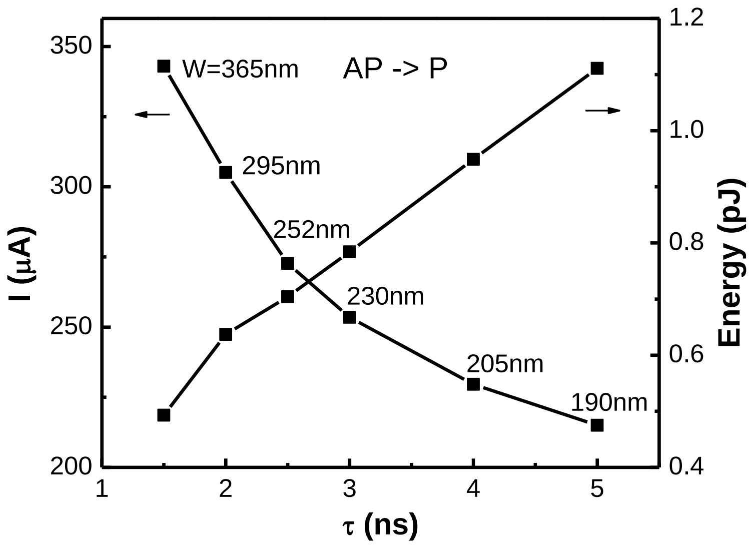


Figure 3.30. Programming current and energy of $AP \rightarrow P$ switching. All points have $2x10^{-5}$ BER.

From Figure 3.29 and Figure 3.30, we see that P $\rightarrow$ AP switching requires much larger current and transistor size in spite of boosted voltage supply (2.2V for P $\rightarrow$ AP and

1V for AP → P). Therefore, the design constraints are set by those of P → AP. Using transistor sizes set by P → AP, AP → P switching will have much lower BER than $2x10^{-5}$, which is not needed and results in a waste of energy. So, we propose to reduce the supply voltage for AP → P switching to save energy.

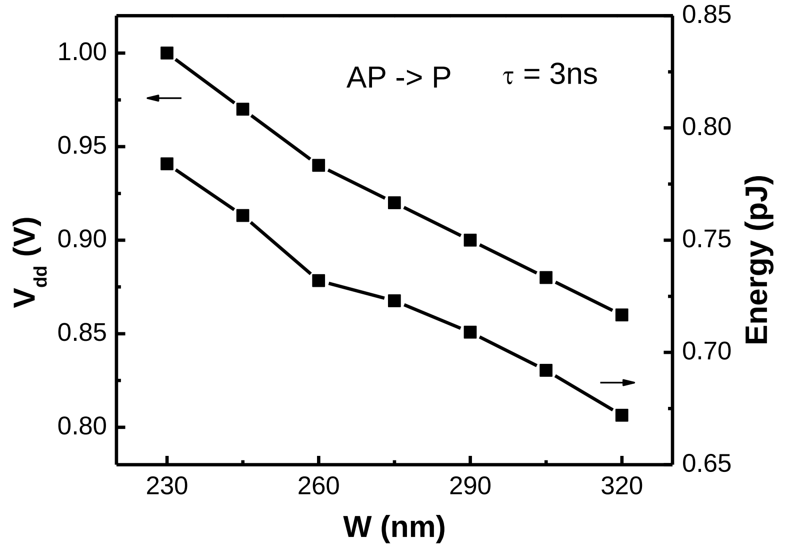


Figure 3.31. Supply voltage and energy at different transistor sizes for *AP → P* switching. All points have BER of $2x10^{-5}$.

Figure 3.31 shows different combinations of supply voltage (Vdd) and transistor width (W) to obtain the same BER of $2x10^{-5}$ for τ = 3ns. The energy decreases with lower Vdd and larger W since current amplitude almost remains the same. As shown in Figure 3.31, when the transistor size increases from 230nm to 320nm (required by P → AP switching), required voltage decreases from 1V to 0.86V to achieve same BER. Thus, programming energy is reduced from 0.784pJ to 0.672pJ (14.3% decrease). However, if the two switchings have equal probability, this optimization causes only 2.3% overall energy reduction. If we make AP $\rightarrow$ P switching more often by some coding techniques, more energy reduction can be achieved.

3.4. Review of Resistive Random Access Memory

Resistive Random Access Memory (RRAM) is another promising emerging memory. It has the advantages of high density, fast read and write speed, good retention and excellent scalability (Wong et al. 2012, 100:1951-1970). RRAM is also capable of multi-level cell (MLC), which is important to be used as the synapses in neural networks (Jo et al. 2010, 10:1297-1301, Yu et al. 2013, 25:1774-1779). RRAM consists of a thin oxide layer sandwiched by two electrodes. It is a polarized device. Depending on the polarity of the voltage pulse on it, a conductive filament can be formed or removed in the oxide layer. The resistance of RRAM depends on the length of the filament or the remaining gap. Longer the filament is, smaller the resistance is. The compact model of RRAM was developed in (Guan et al 2012, 33:1405-1407). The current is an exponential function of the length of the gap and the voltage. And the gap is calculated iteratively with the parameters of the voltage and temperature.

4. Machine Learning Hardware Design

This chapter designs the parallel programming scheme and circuitry for learning on an RRAM cross-point array.

4.1. Introduction

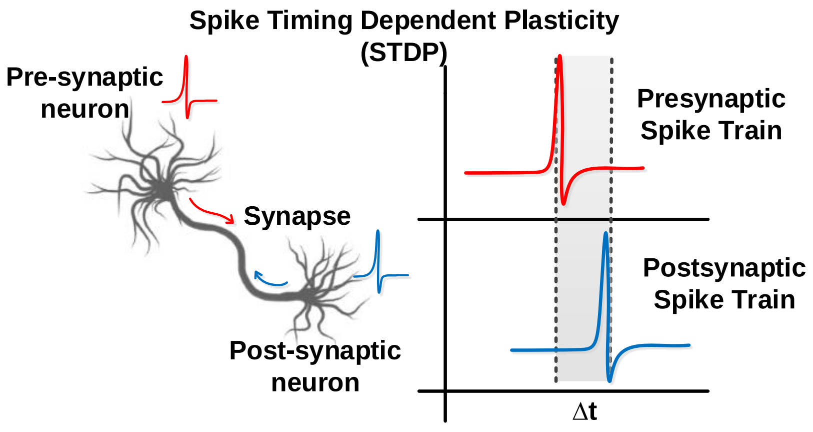
Inspired by the daunting computational capability of the human brain, cognitive computing and learning that are inspired by neuroscience have become an increasingly attractive paradigm for future computation beyond the von Neumann architecture. Along this path toward machine intelligence, learning compact representations on data adaptive dictionaries is the state-of-the-art method for analysing big data (Tosic and Frossard 2011, 28:27-38). It aims to minimize the reconstruction error $\sum_i \| D \cdot Z_i - x_i \|^2$, where x_i is an input vector, D is called the dictionary and Z_i is the coefficient vector which is usually assumed to be sparse in many problems. Such an optimization target is motivated by the sparseness in visual cortex, minimizing both the error and energy consumption in learning.

However, when the data set is big, which is often the case, optimizing the dictionary is a computational challenging problem. Stochastic Gradient Descent (SGD) (Bousquet and Bottou 2008) is one of the most efficient algorithms to solve this problem. Instead of updating the dictionary by full gradient descent, SGD updates the dictionary by using randomly selected gradient as follows:

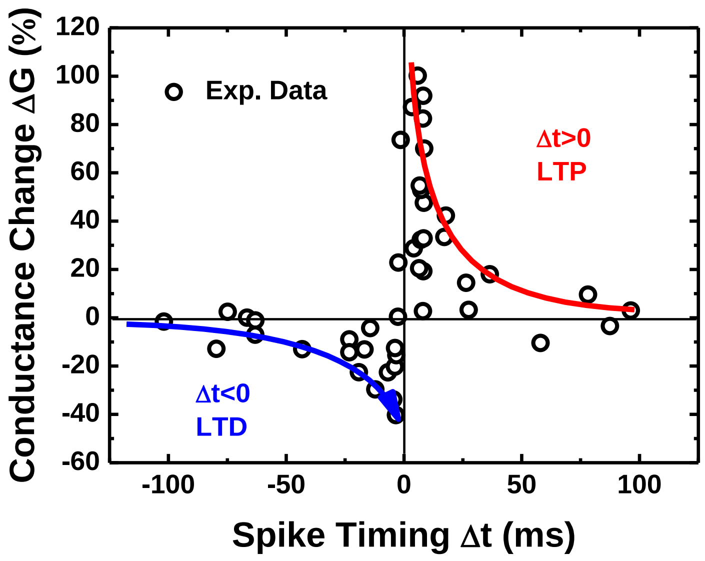
$$D_{t+1} \leftarrow D_t - \eta_t \cdot \Delta D_t, \tag{13}$$

where η_t is the learning rate, $\Delta D_t = r_t \cdot Z_t^T$ and the residual error of data presentation (r) is $r_t = D_t \cdot Z_t - x_t$.

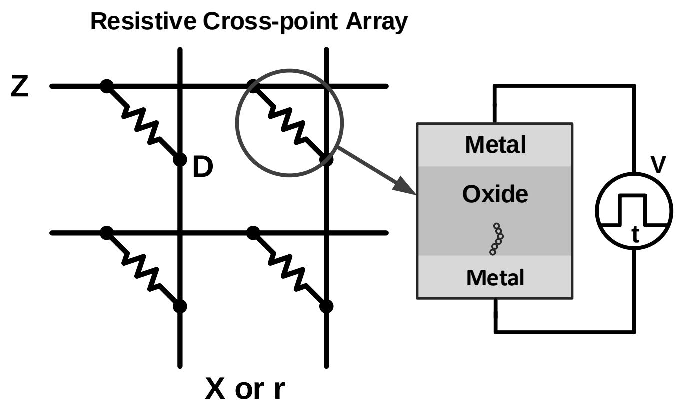
An analogy to this dictionary learning could be found in neural networks in our brain, which consists of spiking neurons and synapses that connect the neurons. During the training process, a spiking neural network *learns* through plastic synapses that change their weights based on the spike timing of the pre-synaptic neuron and the post-synaptic neuron. This learning rule is known as spike-timing-dependent-plasticity (STDP) (Song et al. 2000, Bi and Poo 1998, 18:10464-10472), as illustrated in Figure 4.1 (a) (b).



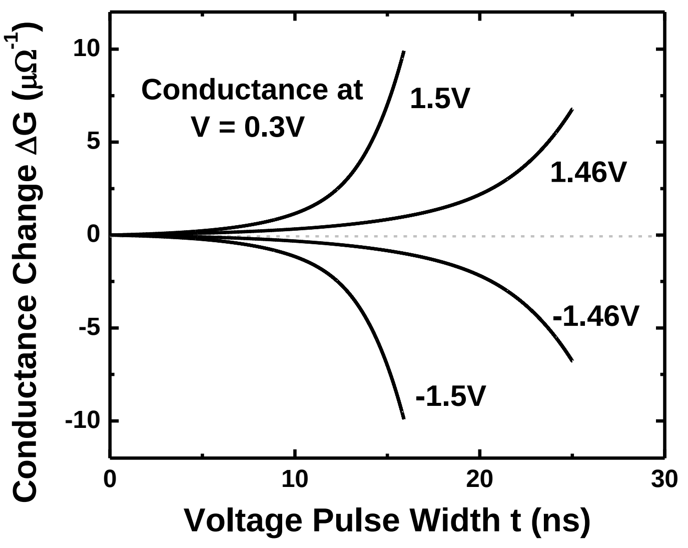
(a)



(b)



(c)



(d)

Figure 4.1. The similarity of programming a biophysical synapse and a RRAM cell. (a) STDP based on the time interval between pre- and post-synaptic spikes. (b) The synaptic conductance change based on STDP. (c) Tuning of RRAM conductance with a voltage pulse across both ends. (d) RRAM conductance change depends on the voltage pulse width.

When these learning algorithms are implemented in hardware to accelerate the learning beyond software limitations, the cross-point array was recently proposed as an effective way to represent synapses with large fan-in and fan-out (Jo et al 2010, 10:1297-

1301, Seo et al. 2011), where each cross-point is implemented with a memory cell. Since scaling conventional on-chip memories (SRAM or eDRAM) becomes more difficult every new technology node, resistive random access memory (RRAM) has emerged as an alternative choice for next-generation memory designs due to its non-volatility, integration density, and low power consumption (Wong et al. 2012, 100:1951-1970).

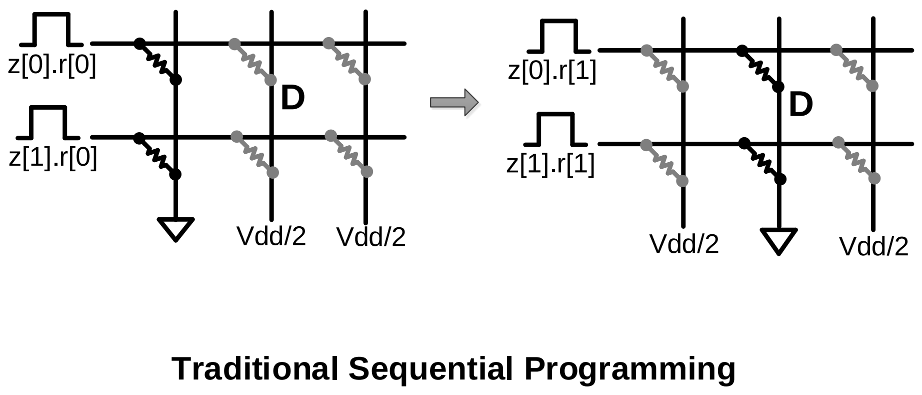
A RRAM cell structure is shown in Figure 4.1 (c), it consists of two metal layers and an oxide layer. The conductance of the oxide layer is determined by the length of the conductive filament. To change the conductance, a voltage pulse needs to be applied across the RRAM cell. Figure 4.1 (d) shows the simulation results on how the RRAM conductance is changed by different voltage pulses. Positive pulses will increase the conductance while the negative pulses will decrease it. It shows that the conductance change is very sensitive to the voltage amplitude and fairly less sensitive to pulse width, which is another reason to use timing to control the programming in fine granularity in this work. We use 1.5V (*Vdd*) as the programming voltage across the two terminals and use 0.75V (*Vdd*/2) to prevent programming.

Using resistive devices for synapses in neuromorphic applications have been actively explored (Jo et al. 2010, 10:1297-1301, Yu et al. 2013, 25:1774-1779). However, updating all the resistive devices in a large cross-point array is still very time-consuming in previous approaches, since it requires sequential operation (row-by-row, column-by-column, or even bit-by-bit). Hereby, we focus on a resistive cross-point array which holds the dictionary values (D), and connects Z (sparse data representation) on one side and r (residual error of data representation on inputs) on the other side. We seek an efficient way to update all the dictionary values stored in a resistive cross-point array by an amount proportional to the multiplication of *Z* and *r* (i.e., $Z \cdot r$). Specific write circuitries are designed for *Z* and *r* on the periphery of the cross-point array, such that the entire resistive cross-point array could be programmed in parallel and thus, the programming speed is not limited by the scale of the dictionary any more.

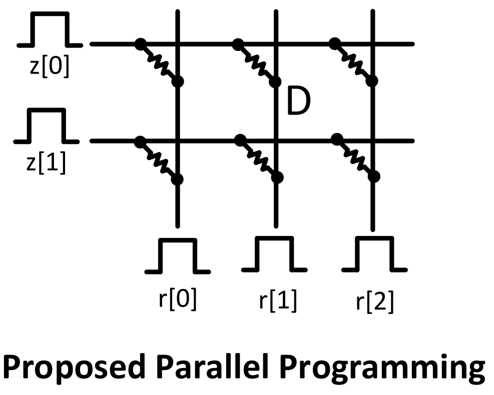
4.2. Parallel Programming Scheme and Circuit Design

Conventionally, programming a resistive memory array is performed sequentially column by column as shown in Figure 4.2 (a), or even bit by bit as usually implemented in the software. In our learning application, to change D value by an amount proportional to $Z \cdot r$, it first needs to calculate $Z \cdot r$ for each column. To program one column of the array, programming pulses that represent the $Z \cdot r$ values of this column are applied on the left side of the array, while this column is connected to ground. The rest of the columns are kept at *Vdd*/2 to prevent programming. After programming one column is finished, the next column can be programmed by applying programming pulses and voltages that correspond to the next column. Therefore, the total time to program the resistive cross-point array using this method is in the order of *O(N)*, where *N* is the number of columns of the array, and its value ranges from 100 to several thousand, depending on the application.

Exploiting the specific property of resistive cross-point arrays that one can simultaneously apply different voltage pulses on each row and column, a parallel programming method is proposed in order to parallelize and accelerate the entire programming process, as illustrated in Figure 4.2 (b). In this method, we do not calculate $Z \cdot r$ before programming, instead pulses that represent *Z* and pulses that represent *r* are simultaneously applied on the rows and on the columns of the cross-point array, respectively. We overlap the *Z* pulses and *r* pulses over the write enable period to effectively realize the multiplication function and thereby increase or decrease the conductance of the RRAM. Specifically, we encode r value into spikes of 1ns pulses in a fixed time period, and encode Z value into the duty cycle of the write period when the r pulses could be applied to each RRAM cell. Thus, in such a synchronous design, the accumulated overlap time of these two pulses in each write cycle indicates the product of $Z \cdot r$.



(a)



(b)

Figure 4.2. The parallel scheme achieves *O(1)* in programming speed, independent on the array dimension.

The write circuit for *Z* generates the programming pulse with a duty cycle proportional to the value of *Z* in a fixed clock period. To program an RRAM cell, the voltage across the cell should be *Vdd*, while *Vdd*/2 is not able to change its conductance. Since the programming voltage of *r* is in the range of 0 to *Vdd*, the effective programming period of the *Z* pulse should supply either 0 or *Vdd* voltage, and the rest should be *Vdd*/2.

Z is always a positive number while *r* can be positive or negative, depending on the residual error. Therefore, whether *D* will increase or decrease depends on the sign of *r*, but not *Z*. When *r* is positive, *D* decreases, and vice versa. Since we don't calculate $Z \cdot r$ up front, the programming voltage of *Z* has to prepare for both positive *r* and negative *r*. In our synchronous design, we divide the write period into two phases, controlled by the clock. The first phase deals with the condition of $r > 0$ (positive period), and the second phase deals with the condition of $r < 0$ (negative period). In the positive period, the effective programming voltage is 0 in a certain time proportional to *Z*. After this time, the programming voltage switches to *Vdd*/2 to prevent further programming. Similarly, in the negative period, the effective programming voltage is *Vdd*, and then the voltage switches to *Vdd*/2. To program a RRAM cell, we can keep the voltage of *r* as *Vdd* in the positive period (for $r > 0$) and as 0 in the negative period (for $r < 0$). Consequently, the voltage across the RRAM cell during the overlap time of *Z* and *r* will be –*Vdd* and *Vdd* for $r > 0$ and $r < 0$, respectively. Such a voltage overlap serves as the basis to tune the RRAM conductance for *D*.

To generate such a pulse pattern, a digital circuit is designed, as shown in Figure 4.3. The inputs include *Z* [15:0], WE, PN and clock. *Z* [15:0] is a pre-decoded natural number, representing the value of *Z* from 0 to 16 by the number of '1' in these 16 bits. The '1's are all sequentially on the right side of *Z* [15:0]. WE is the global control write enable signal.

The writing is performed when WE = 1. PN is the signal that differs the positive period and negative period. PN = 0 means positive period and PN = 1 means negative period. The clock signal is an internal clock. There are 32 cycles in the whole write period.

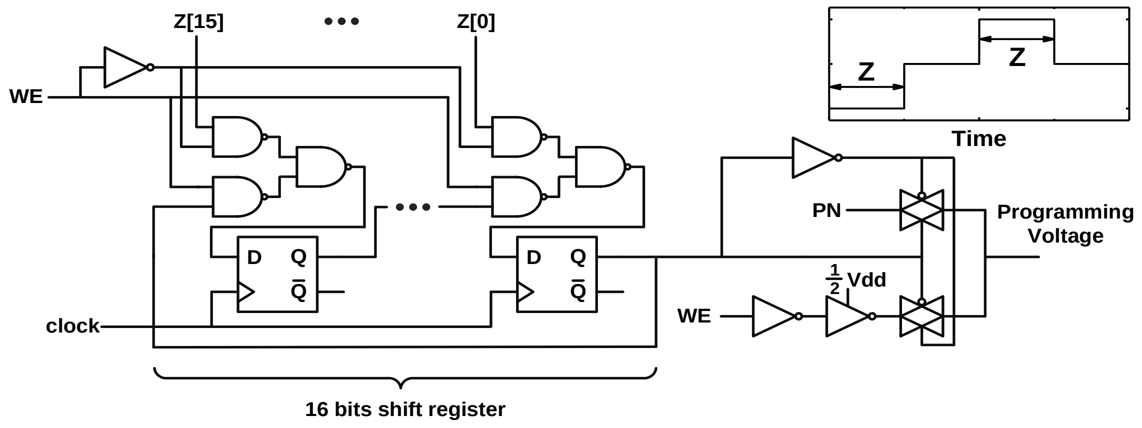


Figure 4.3. Circuit schematic to generate the programming voltage *Z*. The inset illustrates the pulse pattern for both phases.

In Figure 4.3, the left part of the circuit is a 16-bit shift register. It converts the parallel input *Z* [15:0] into a sequential output. Thus, the time when the output is 1 is proportional to the value of *Z*. Note that the output of the shift register is connected back to the first stage of itself in order to recycle the data *Z*. With 32 clock cycles for one write period, the shift register generates two identical pulses with the duty cycle proportional to the value of *Z*. These two identical pulses are further input to the mux to generate different programming voltage levels for the positive period and the negative period, which is controlled by both WE and PN. With the whole circuit above, we are able to convert the value of *Z* into the duty cycle of pulses for both cases of $r > 0$ and $r < 0$.

The write circuit for r generates a train of pulses, where (1) the number of pulses is proportional to the value of *r*, (2) each pulse has a fixed width (for fixed RRAM programming period) and (3) the pulses are evenly distributed across a constant write period.

Whenever there is overlap between the *Z* window and an *r* pulse, the fixed pulse width ensures that the conductance of RRAM is changed by a fixed amount. The uniform distribution of pulses is important to minimize the quantization error in our method, which effectively multiplies *Z* and *r*. *r* could be a positive or negative value, which would increase or decrease the RRAM conductance, respectively. Since the required voltage values for increasing and decreasing the RRAM conductance are different, each write cycle was separated into two phases, where the first phase generates signals for positive *r* values and the second phase for negative *r* values.

In order to increase the resistance of the RRAM, a positive voltage of *Vdd* is required between the Z and R nodes. Thus in the first phase, if *r* is positive, active-high pulses (number of pulses proportional to *r*) are generated with a fixed pulse width of 1ns, while *Z* is driven low (the time at low is proportional to *Z* value). Through this operation in the first phase, a fixed voltage ($V_R – V_Z$ = 1.5V) is applied to each RRAM cell for the accumulated overlap time that represents $Z \cdot r$. If *r* is negative, the output signal is kept at low during the first phase, ensuring no change in the resistance of the RRAM cells. Similarly, in the second phase, if *r* is positive, the output is kept at high to ensure no change in resistance of the RRAM cells. On the other hand, if *r* has a negative value, then in the second phase active-low pulses are generated with a fixed pulse width of 1ns while *Z* is driven high. Thus, a fixed voltage in the opposite direction in the case of positive *r* value ($V_Z – V_R$ = 1.5V) is applied to the RRAM cells for the accumulated overlap time that represents $Z \cdot r$. After each write cycle, the RRAM conductance will increase or decrease by an amount proportional to $Z \cdot r$.

The circuit implementation consists of various delay elements forming a configurable ring oscillator (RO) with a start and polarity control. Write Enable (WE) and sign-bit of *r* determine the phase in which the pulses are to be generated and their polarity. The number of pulses during the write period is varied by changing the length of the ring oscillator and thus, its frequency. This was achieved using switches, which determines the total gate delay in the ring oscillator. The control of the switches is generated from the *r* value, ensuring that only one switch is on for a particular value of *r*. When $r = 0$, no change in the RRAM conductance is allowed. In total, 15 buffer stages (d_1-d_{15}) in Figure 4.4 are implemented with different delay values, such that the number of pulses generated for each write cycle is proportional to the *r* value. The fixed pulse width (1ns) is generated after each rising edge of the RO output. Based on the sign-bit of *r* and the write phase (PN), the final mux stages select among *Vdd*, 0, pulse generator output or the inversion of pulse generator output.

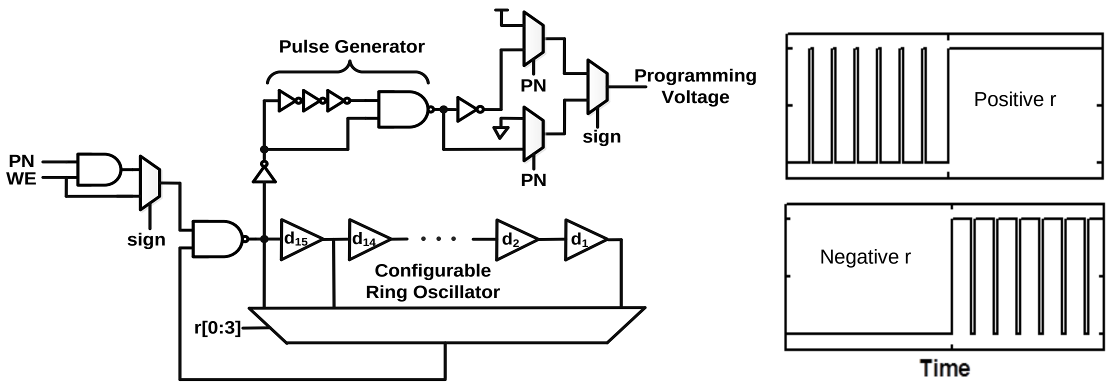
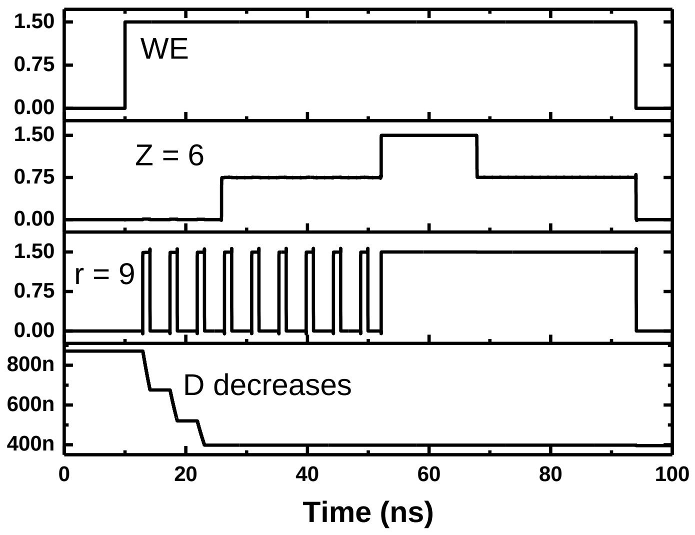


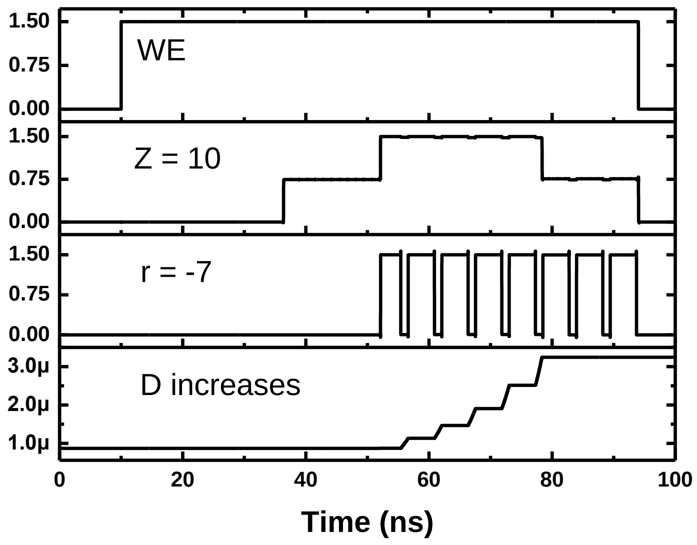
Figure 4.4. Circuit schematic to generate the programming pulses of *r*.

4.3. Simulation and Results

In this section, we show the simulation results of the overall system that consists of parallel programming circuits and RRAM cells. The write circuitries for *Z* and *r* are implemented in 65nm CMOS technology, and we used a spice model for the RRAM device that was generated from device measurements.



(a)



(b)

Figure 4.5. Timing diagram of the programming system. Through the overlap in time between *Z* and *r* pulses, it demonstrates that (a) *D* decreases when $r > 0$. (b) *D* increases when $r < 0$.

Figure 4.5 shows the timing diagram of the parallel programming system. When the write enable (WE) signal turns on, both *Z* and *r* write circuitries start pulse programming based on the values of *Z* and *r*, and thus change the value of *D* during the overlap time of the two pulses. In Figure 4.5 (a), it is shown that when *r* is positive, the programming occurs in the first half of the write period and the value of *D* decreases. Figure 4.5 (b) illustrates that when *r* is negative, the programming happens in the second half of the write period and the value of *D* increases. Independent of the array size, the parallel programming of the entire array requires only 84 ns, while the sequential programming requires 1.6 μs for a 400 x 100 array. The simulation also shows that the energy consumption of parallel programming of the 400 x 100 array is about 13.9 nJ. The layout areas of Z and r circuitries are 850 μm^2 and 1154 μm^2, respectively.

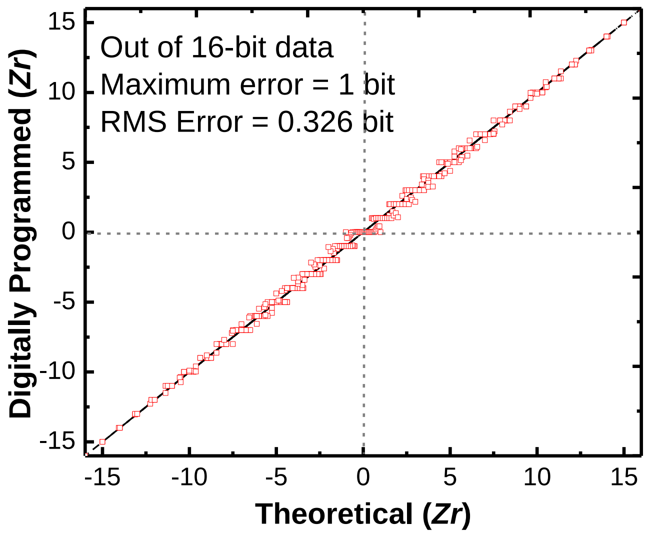


Figure 4.6. The quantization error of the parallel programming method, with the maximum error at 1 bit (6.25%).

The method of using overlap time of *Z* and *r* pulses with a certain granularity to calculate multiplication introduces quantization error. To analyze this, we performed simulation for all combinations of *Z* and *r* values (both from 0 to 16). Figure 4.6 shows the comparison of the simulated results of $Z \cdot r$, namely the accumulated overlap time of *Z* and *r* pulses, to an ideal $Z \cdot r$ multiplication. It is observed that the digital programming mostly follows the ideal multiplication closely, while producing the maximum error of 1 bit (out of 16 bits) when both *Z* and *r* are small.

We compared the proposed system against a software implementation on the task of updating the dictionary *D*. For this purpose, we used MNIST dataset (LeCun et al. 1998, 86:2278-2324) to extract the image feature with Stochastic Gradient Descent (Bousquet and Bottou 2008) algorithm. For the software approach, we used Intel Core i5 2.4 GHz dual-core processor and 4 GB memory. Figure 4.7 shows the dictionary *D* before and after the feature extraction. The computation time consumed to update *D* for this entire dictionary learning process is 750 μs per image patch (10 iterations). With our proposed hardware approach, a 400 x 100 resistive cross-point array is used to achieve the computation time of 840 ns per image patch, which is a 900X improvement over the software implementation for the identical dictionary learning.

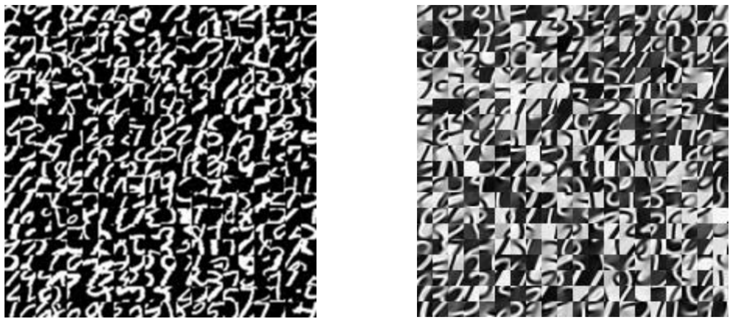


Figure 4.7. Demonstration of the proposed method in updating the dictionary. Current software approach: Processor: Intel Core i5 2.4GHz 2 cores; Memory: 4 GB; Computing time: 750 μs per image patch. Proposed parallel programming hardware approach: RRAM array dimension: 400 x 100; Computing time: 840 ns per image patch.

5. Bio-Inspired Learning Algorithm

To realize learning on-a-chip, optimization of the learning algorithm is required. Inspired by the fact that animal brains feature high accuracy, efficient energy use, and low hardware overhead, this work develops a new learning algorithm based on biologically plausible learning rules to improve the efficiency in sparse learning.

5.1. Introduction

Neuro-inspired computing, including learning and inference, has made significant progress in recent years and will fundamentally alter the way individuals and organizations live, work and interact with each other (Schmidhuber 2015, 61:85-117, LeCun et al. 2015, 521:436-444, Furber 2016 13:051001, Liu et al. 2017, 234:11-26). Machine learning and deep learning algorithms have been successfully applied to many data processing and analysis tasks, including feature extraction from images and videos (Hong et al. 2015, 24:5659-5670, Yu et al. 2016, 99:1-11, Yu et al. 2017, 12:1005-1016, Zhang et al 2017), image segmentation (Pan et al. 2017, 229:88-99), and big multimedia analysis (Yu et al. 2017). While many previous efforts have been made to improve the optimization algorithms for artificial neural networks (Sun et al. 2017, 230:374-381, Baig 2017, Scardapane 2017, 241:81-89), the computational complexity of artificial neural networks still challenges the state-of-the-art hardware platforms, especially mobile applications that are tightly constrained by energy efficiency and hardware size (Lane and Georgiev 2015). In contrast, animal brains, as a natural system for information processing, exhibit extraordinary features of ultra-high energy efficiency (Sarpeshkar 1998, 10:1601-1638), low hardware overhead, and high accuracy in perceptual and learning tasks. For instance, the olfactory system in fruit flies only contains about 5000 neurons (Galizia and Sachse 2010); after a very small number of stimulus presentations, it is capable to detect tens of thousands of odors at very high accuracy. The locust antennal lobe consists of ~830 excitatory projection neurons and 300 inhibitory local neurons (Emst et al. 1977, 176:285-308), achieving sparse odor representation that is specific over thousand-fold changes in odor concentration (Stopfer et al. 2003, 39:991-1004). Indeed, the efficiency of information processing by the sensory and cortical systems is vitally important to animal survival in nature.

Many efforts have been made to capture the advantages of the nervous systems by creating computational models, with biologically plausible learning rules (Perez-Orive et al. 2004, 24:6037-6047, Assisi et al. 2007, 10:1176-1184, Huerta and Nowotny 2009, 21:2123-2151, Zylberberg et al. 2011, 7:1-12, King et al. 2013, 33:5475-5485, Querlioz et al. 2013, 12:288-295, Diehl and Cook 2015). Among neurons, the two basic forms of data transmission are excitation and inhibition. Excitation has been extensively shown to be the primary path of data processing and feature extraction. Neural network models with excitation only can be trained to recognize images, differentiate objects, and categorize input data (Huerta and Nowotny 2009, 21:2123-2151). In addition to excitation, the inhibition provided by interneurons is indispensable to learning and behavioral adaptation, as observed in a variety of species (Kelsom and Lu 2013, 3:1-19). Even though the number of inhibitory interneurons is usually much smaller than that of the excitatory ones, the chemical blockade of the inhibition path results in pronounced deficits in decision making, recognition or memory recall (Sillito 1977, 271:699-720, Tsumoto et al. 1979, 34:351-363, Sawaguchi et al. 1996, 75:2150-2156, Perez-Orive et al. 2004, 24:6037-6047).

The combination of excitatory and inhibitory paths forms the network element, which is further connected into the complex nervous system (Assisi et al. 2007, 10:1176-1184).

Systematic investigations (Sporns and Kötter 2004, 2:1910-1918, Alon 2007, 8:450-461) have identified a small set of recurring structural elements, called neural motifs, serving as efficient building blocks to realize diverse functions. Depending on the interaction among neurons, there are two common inhibitory motifs: feedforward (FF) and feedback (FB). In the FB (or recurrent) inhibitory motif, the inhibitory cells are driven by a population of excitatory neurons and in turn inhibits the same population of the excitatory cells. FF inhibition occurs between different brain areas, where the inhibition cells receive the signal from the excitation group and then act on a different group of postsynaptic excitatory neurons (Assisi et al. 2007, 10:1176-1184). Both FB and FF inhibitory motifs can limit the firing of the postsynaptic neurons and facilitate the construction of sparse representation from input data. While FB inhibition provides dynamic thresholding in the learning process (Masson et al. 2002, 417:854-858), FF inhibition is vital to maintaining the firing rate and sparse representation across a wide range of input conditions. For instance, external stimuli can vary in strength by many orders of magnitude and yet the FF network still faithfully represents the stimulus with little change in the response (Pouille et al. 2009, 12:1577-1585). Moreover, the FF inhibitory path speeds up the response time and creates relatively fast inhibition (Alon 2007, 8:450-461). In fact, the rapid reaction induced by the FF inhibitory motif has also been found in many other biological networks, such as transcription regulation in the gene system (Bahrami and Drabløs 2016, 62:37-49).

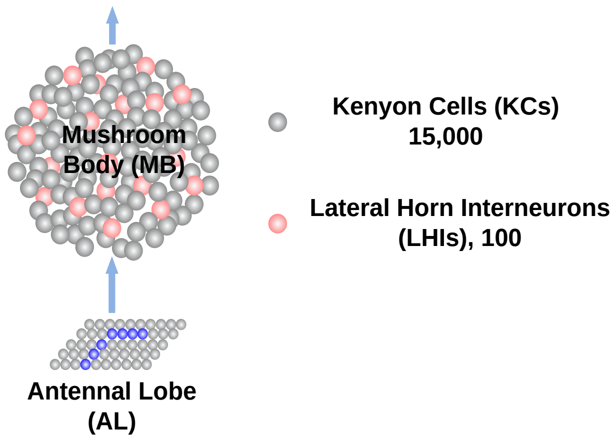
Most prior computational models with inhibition were on lateral inhibition among the excitatory cells or on the FB path in training and inference. For example, the SAILnet model was able to perform pattern learning with direct inhibitory connections between the excitatory neurons (Zylberberg et al. 2011, 7:1-12); an extension of the sparse coding algorithm leveraged lateral inhibition to form the dictionary in training (Szlam et al. 2011). Yet such lateral inhibition between the excitatory cells is not consistent with the physiological properties of primary visual cortex, in which the inhibitory interneurons (GABAergic) are separated from the excitatory neurons (Kelsom and Lu 2013, 3:1-19). The work of E-I Net advanced the model with a group of FB inhibitory neurons, which enable sparse coding by actively de-correlating the excitatory population (King et al. 2013, 33:5475-5485). Meanwhile, the knowledge of FF inhibition has been accumulated from both biological and computational perspectives. Multiple formats of inhibitory plasticity were summarized in (Vogels et al. 2013). Haider et al. (2006, 26:4535-4545) demonstrated the importance of the balance between the excitation and inhibition paths. Skorheim et al. (2014, 9:1-15) built a computational model with both feedforward excitation and inhibition that mimics the olfactory system of insects for specific foraging task.

Inspired by these results, this work develops a general spiking neural network model with FF inhibition to illustrate its critical role in cognitive learning and hardware efficiency, as shown in Figure 5.1. Using biologically plausible rules for plastic synapses and spiking neurons, this new model serves as the testbed to analyze the impact of multiple factors on learning accuracy and speed. The results further help shed light on energy-efficient learning algorithms that may benefit from the FF inhibitory motif. The contributions of this work are summarized as the following:

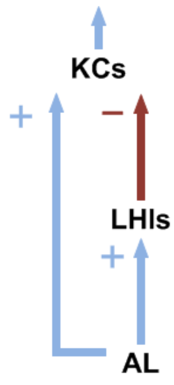
A biologically plausible spiking neural network is proposed, with the feedforward inhibitory motif that is inspired by the olfactory system of insects.

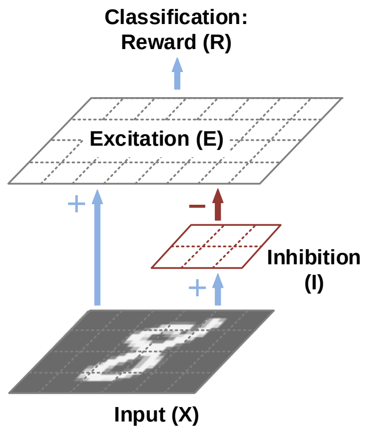
The new network model achieves more than 3X reduction in the network size, compared to state-of-the-art biologically plausible spiking neural network at the same accuracy of 95% in the recognition task of handwritten digits.

The role of feedforward inhibition in sparse learning, as well as associated parameters, is systematically studied. Its function of coarse categorization is confirmed.


Mushroom Body (MB)
Kenyon Cells (KCs) 15,000
Lateral Horn Interneurons (LHIs), 100
Antennal Lobe (AL)

(a)





(b) (c)

Figure 5.1. The feedforward inhibitory motif and the structure of the inspired neural network model. (a) The computational model of the olfactory system of insects (Yu et al. 2017). Mushroom Body (MB) receives input from Antennal Lobe (AL). The majority part

of MB consists of a large number of Kenyon Cells (KCs), while Lateral Horn Interneurons (LHIs) only contributes to a small fraction of MB. LHIs receive the same input as KCs and generate feedforward inhibition that suppresses the firing of KCs. (b) The feedforward inhibitory motif extracted from the olfactory model. (c) The spiking neural network model based on the feedforward inhibitory motif. It is trained for the handwritten digits recognition task. The input layer is a full 28x28 image containing a number between '0' and '9'. The excitation layer (E), similar as the KCs in MB, has a large number of excitatory neurons. The inhibition layer (I), similar as the LHIs in MB, has a small number of inhibitory neurons that suppress the neuron firing in E layer.

5.2. Feedforward Inhibition Spiking Neural Network

To illustrate the advantage of a biological nervous system, we construct a neural network model that has the following important features: at the device level, it should only use biologically plausible rules for both synaptic plasticity and spiking neurons, instead of artificial rules; at the system level, it should achieve sufficiently high accuracy in learning compared to other artificial algorithms. Under these constraints, we explore the structure, function and computation efficiency of the network model, with the focus on the feedforward inhibitory motif.

Figure 5.1 (a) and (b) present the structure of the feedforward inhibitory motif (FFI). It is inspired by the insect brain, especially the olfactory system. In such a system, the plasticity is located in the Mushroom Body (MB) to process input signals and accomplish fast and efficient classification. While the number of neurons and their physiological connectivity in the MB are well studied (Perez-Orive et al. 2004, 24:6037-6047), the exact functional map and the learning mechanism remain as open questions. One of the characteristics of the MB is the existence of a FFI path that suppresses the main excitation path to generate the output. Even though FFI only consists of a small portion in the MB, experiment results confirmed its critical role to the formation of sparsity during the training process (Perez-Orive et al. 2004, 24:6037-6047). We argue that the function of FFI is to effectively promote fast and reliable learning, as well as the reduction of network size at high learning accuracy. This work aims to substantiate such claim by testing the FFI motif with the MNIST database of handwritten digits (LeCun et al. 1998, 86:2278-2324). Figure 5.1 (c) presents the proposed network structure, based on the feedforward inhibitory motif. It consists of four layers: the input layer, the excitation layer (E), the inhibition layer (I), and the classification layer (C). The inhibition layer functions as feedforward inhibition. It receives innervation from the input layer and sends out inhibitory pulses to the excitation layer. The excitation layer receives innervation from the input layer and receives inhibition from the inhibition layer. It extracts the features of input. The classification layer receives innervation from the excitation layer and does the classification based on the features. The size of each layer depends on the application task. The following subsections present the spiking neuron model, learning rules, balancing method and the training procedure.

5.2.1. Spiking Neuron Model

At the device level, the neuron model used in this paper is the leaky-integrate-fire (LIF) model described in population (King et al. 2013, 33:5475-5485). The membrane potential of each neuron is initially reset to 0 and then accumulates by the weighted sum of all input signals. It increases or decreases by a certain amount, depending on whether the input signal is from the excitatory or inhibitory synapse, respectively. In addition, the membrane potential decays at a rate proportional to itself during the integration stage.

When the membrane potential is larger than the threshold, the neuron fires a spike out, and the membrane potential is reset to 0. This model is described by the following equations.

$$u_i(t+1) = u_i(t)\exp\left(-\frac{1}{\tau}\right) + \sum_j \beta_j z_j(t) W_{ij} \tag{13}$$

$$z_i(t+1) = \begin{cases} 1, & u_i(t+1) \geq \theta_i \\ 0, & u_i(t+1) < \theta_i \end{cases} \tag{14}$$

$$u_i(t+1) = 0 \quad if \quad z_i(t+1) = 1 \tag{15}$$

$u_i(t)$ is the membrane potential of neuron i at time t. τ is the membrane time constant governing membrane potential decay rate, with unit of number of simulation time step. $z_i(t)$ is the spike output of neuron i at time t, which is either 1 for spike or 0 for no spike. W_{ij} is the synaptic weight from neuron j to neuron i. β_j indicates the type of input neuron j, which is either 1 for the excitatory input and -1 for the inhibitory input. θ_i is the threshold of neuron i. As the full image is used as the input, as shown in Figure 5.1 (c), each input neuron represents one pixel in the original image, which generates excitatory signals. Each image is presented for a time window of 50 simulation time steps. The spike rate is calculated within this time window. So, the minimum precision of the spike rate is equivalent to 0.02.

5.2.2. Learning Rules of Synaptic Plasticity

From the input image, the FF neural network generates the sparse feature, which is represented by the spike rate of the E layer; then the classifier produces the classification score, i.e., the reward signal (R). If the prediction of the classification layer (C) matches the label of the input image, the reward is 1, otherwise, the reward is -1. Based on the digitalized reward signal, the rewarded Spike Rate Dependent Plasticity (SRDP) is applied to update the synaptic weight; this is the rewarded training scheme. In case of unsupervised training, there is no reward term in the following equations for SRDP. For excitatory synapses, e.g. X to E and X to I, they follow the Hebbian rule:

$$\Delta W\_XE_{ij} = \eta\_XE_{ij} R X_i E_j \tag{16}$$

$$\Delta W\_XI_{ij} = \eta\_XI_{ij} R X_i I_j \tag{17}$$

X_i is the value of the i-th input neuron X. E_j is the spike rate of the j-th neuron E. I_j is the spike rate of the j-th neuron I. ΔW_XE_{ij} is the change of the synaptic weight between neuron X_i and E_j. ΔW_XI_{ij} is the change of the synaptic weight between neuron X_i and I_j. R is the reward signal corresponding to one input image X. η_XE_{ij} is the learning rate of the synapse from neuron X_i to E_j. η_XI_{ij} is the learning rate of the synapse from neuron X_i to I_j. All the learning rates decay with the increasing number of update of each synapse, modeled by:

$$\eta_{ij} = \frac{\eta_0}{\alpha + N_{ij}} \tag{18}$$

η_0 is the constant base learning rate. α is a constant called habituation rate, governing the speed of decay of the learning rate. Large α indicates the learning rate decays slowly with the number of updates and small α indicates the learning rate decays fast with the number of updates. N_{ij} is the number of all the previous updates of the synapse from neuron i to neuron j. This decay of the learning rate is called local habituation. The learning rate of each synapse only depends on the update history of the synapse itself. Thus, local habituation is biologically plausible. The decay of learning rate can stabilize the training and guarantee convergence.

The inhibitory synapses between E and I have slightly different update rules:

$$\Delta W\_IE_{ij} = \begin{cases} \eta R I_i E_j, & E_j < \rho_{weak} \\ \eta R I_i (E_j - \rho_{target}), & E_j > \rho_{strong} \end{cases} \quad (19)$$

ΔW_IE_{ij} is the change of the synaptic weight between neuron I_i and E_j. I_i is the spike rate of the i-th I neuron. E_j is the spike rate of the j-th E neuron. ρ_{weak} is the upper bound of weak spiking rate; ρ_{strong} is the lower bound of strong spike rate. ρ_{target} is the target spike rate for the strongly active neurons, $\rho_{low} < \rho_{high} < \rho_{target}$. This rule is inspired from (Vogels et al. 2013). Strongly active neurons receive less inhibition and weakly active neurons receive more inhibition. Thus, the contrast of population response is enhanced.

In our SRDP rules, the update of each synaptic weight only depends on the activity of its pre-neuron and post-neuron, and a global reward signal. No backpropagation is involved. Our SRDP rules are biologically plausible.

5.2.3. Balancing Methods

Since Hebbian learning is not balanced, we applied two balancing methods in supplement to the synaptic plasticity, input balancing and homeostatic balancing as used in the paper by Skorheim et al. (2014, 9:1-15). Input balancing is to ensure that the total synaptic weight of an input neuron remains unaffected by individual plasticity event. A scaling process is implemented after each SRDP event. When the weight of a synapse increases or decreases, the weight of all the other synapses connected to the same input neuron decreases or increases by a scaling factor.

$$W_{ij(n+1)} = W_{ij(n)} \frac{W_{j0}}{\sum_i W_{ij(n)}} \quad (20)$$

$W_{ij(n)}$ are synaptic weights after SRDP update, but before the balancing. $W_{ij(n+1)}$ are synaptic weights after balancing. W_{j0} is the total synaptic weight of input neuron j. The input balancing can prevent one input neuron from dominating the output or being ignored by the output neurons. The homeostatic balancing is to ensure that all the output neurons maintain a relatively constant long-term firing rate. It is applied once after every batch of 100 training images. All the synaptic weights of an output neuron will increase or decrease if the firing rate of this neuron is low or high, respectively.

$$\Delta W_{ij} \propto -\left(Sr_j - Sr_{avg}\right) \tag{21}$$

Sr_j is the long-term spike rate of the j-th output neuron for a batch of images. Sr_{avg} is the average spike rate of a batch of input samples and of all the neurons in the same layer. The homeostatic balancing can improve the neuron utilization.

5.2.4. Training Procedure

Algorithm 1 Training procedure of the rewarded SRDP of FFI SNN

```
Initialize all the synaptic weights
For input images = 100 images in training dataset
Feedforward through the network for 50 time steps
Calculate the spike rate for E and I neurons
Calculate C, update the classification layer W_EC and calculate the reward
Update the excitatory synaptic weights W_XE and W_XI with reward, and apply the input balancing
Feedforward through the network for 50 time steps
Calculate the spike rate for E and I neurons
Calculate C, update the classification layer W_EC and calculate the reward
Update the inhibitory synaptic weights W_IE with reward, and apply the input balancing
Apply homeostatic balancing
If number of batches is 50, 100, 150…
   Feedforward through the network with previous 50 batches of images
   Calculate the spike rate for E and I neurons
   Calculate C, update the classification layer W_EC
End If
End For when all training images are used or stopping criterion is met
```

The rewarded training procedure is described by Algorithm 1. The weights of excitatory synapses in the network are initialized by random values uniformly distributed between 0 and 1. Each neuron in the excitation layer and the inhibition layer only randomly connects to 50% of input neurons during the initialization. Such 50% connectivity mimics the olfactory system (Assisi et al. 2007). The connectivity of a specific synapse remains the same after the initialization. The training is mini batch based and separated for excitatory synapses and inhibitory synapses. For a batch of 100 training images, we first simulate the feedforward of the network and calculate the spike rate of all neurons for every image. For rewarded training, the output of classification layer (C) and the reward (R) are then calculated. Each neuron in the C layer represents one category in the task. Each C neuron computes the weighted sum of the spike rate of E neurons and then is normalized by the sum of the entire layer. The neuron with the maximum output is the prediction. If the prediction matches the label of the input image, the reward is 1, otherwise, the reward is -1. The weights between E and C layer are updated only when the reward is -1 with the following rule:

$$\Delta W\_EC_{ij} \propto -E_i(C_j - C_{th}) \tag{22}$$

ΔW_EC_{ij} is the change of the synaptic weight between neuron E_i and C_j. E_i is the spike rate of the i-th E neuron. C_j is the output of the j-th C neuron. C_{th} is a constant to differentiate strong C and week C, and is tuned to be 0.105 for the optimal classification accuracy. This rule is a punishment only Hebbian learning rule variant, which can maximize the prediction accuracy of the classification layer. After updating the classification layer, the reward is also updated. Next, the excitatory synaptic weights are updated from the input layer to the excitation and inhibition layers. With updated excitatory synaptic weights, the FF network is simulated once again for this batch to update the weights from the excitation to the classification layer, as well as the inhibitory synaptic weights. The separation of update excitatory synapses and inhibitory synapses is for more stabilization and easier tuning of this highly non-linear system. The input balancing is applied after every update of synaptic weights. Homeostatic balancing is applied once after training one batch. After this step, the training moves on to the next batch. Every 50 batches, the classification layer is trained once again with all the 50 batches together through the FF network, to further improve the performance of the network.

For unsupervised training, we don't need to train the classification layer every batch, because the reward is not needed. The classification layer is only trained after every 50 batches, using all the 50 batches together through the FF network. The unsupervised training procedure is described by Algorithm 2.

Algorithm 2 Training procedure of the unsupervised SRDP of FFI SNN

```
Initialize all the synaptic weights
For input images = 100 images in training dataset
Feedforward through the network for 50 time steps
Calculate the spike rate for E and I neurons
Update the excitatory synaptic weights W_XE and W_XI without reward, and apply the input balancing
Feedforward through the network for 50 time steps
Calculate the spike rate for E and I neurons
Update the inhibitory synaptic weights W_IE without reward, and apply the input balancing
Apply homeostatic balancing
If number of batches is 50, 100, 150…
   Feedforward through the network with previous 50 batches of images
   Calculate the spike rate for E and I neurons
   Calculate C, update the classification layer W_EC
End If
End For when all training images are used or stopping criterion is met
```

5.3. Results and Discussion

A neural network model is built with the feedforward inhibitory motif, and leaky integrate-and-fire neurons as shown in Figure 5.1 (c). The spike rate in a window of 50 time steps is used to represent the data. The membrane time constant τ is set to 10. A Hebbian Spike-Rate-Dependent-Plasticity (SRDP) is applied to all synapses. No backpropagation is involved in the training. The base learning rate η_0 is 3 for the excitatory synapses and 1 for the inhibitory synapses to balance the amount of weight change of excitatory and inhibitory synapses. The habituation rate α is 50 for the excitatory synapses and 100 for the inhibitory synapses. As a proof of concept, this network is trained with a representative machine learning benchmark, the MNIST dataset, for the classification of handwritten digits. This dataset consists of 60000 training images and 10000 test images. Each image is a 28x28 grey scale (256 intensities) image of a single numerical digit between '0' and '9'. We use the full image as the input to the network. To categorize these images, we use 2000 excitatory neurons (E) and 100 interneurons (I). Both excitation and inhibition layers receive innervation from half of the input neurons that are randomly selected. Each excitatory neuron receives innervation from all the interneurons. For the classification layer, it receives the input from all the excitatory neurons. There are 10 neurons (C) in the classification layer, representing 10 classes from ‘0’ to ‘9’. Their spike rate represents the probability that an image belongs to each class. The prediction is the class with the maximum probability. More details on modeling and training can be found in Section 5.2.

5.3.1. Performance of the Feedforward Inhibitory Motif

After training with 60000 training images, the network achieved 95.0% classification accuracy on the testing set. Figure 5.2 shows the firing map of 2000 E neurons corresponding to 1000 images before and after training. The images are randomly selected from the training set and sorted by their true labels. Each point in the map represents the firing rate of one E neuron for one input image. Each row is for one E neuron and each column is for one image. The firing map after training presents a good distinction of different classes. The images of the same class have similar firing patterns. As a comparison, in the initial firing map before training, many neurons tend to fire for a wide range of classes and thus, fail to distinguish different classes.

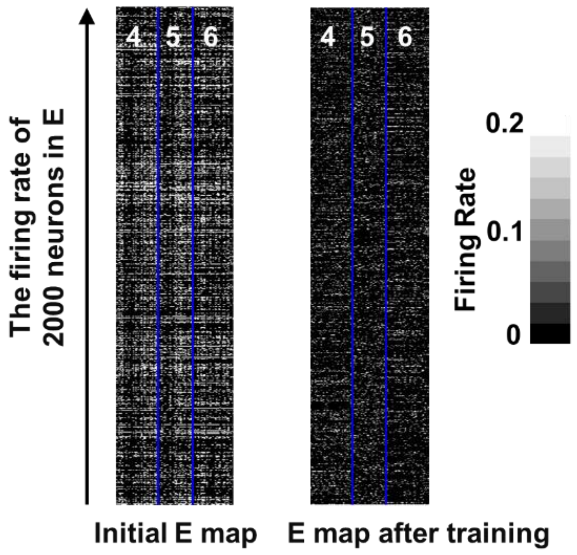


Figure 5.2. The firing map of 2000 excitatory neurons for 1000 images, before and after training. The images are randomly selected from the training set and sorted by their true labels (number 4, 5, 6). Each point in the map represents the firing rate of one E neuron for one input image based on grayscale. Each row is for one E neuron and each column is for one image.

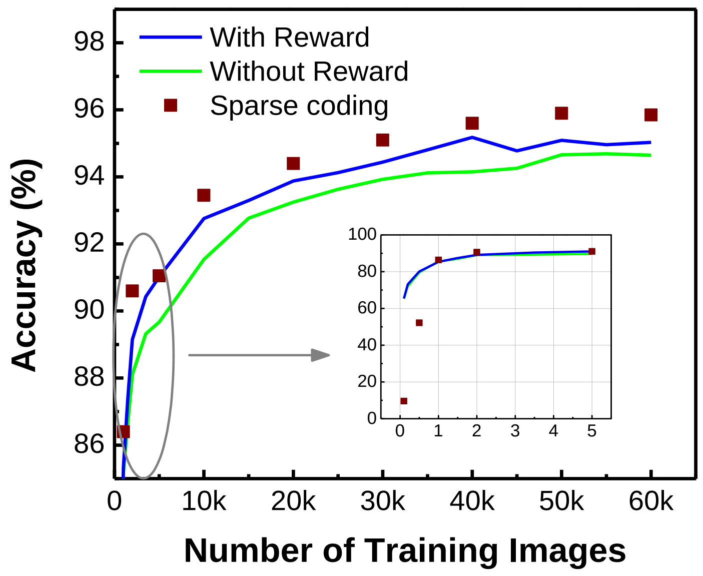


Figure 5.3. Testing accuracy as a function of the number of training images, in both rewarded and unsupervised (without reward) training, compared with the sparse coding algorithm that uses 6-bit data and stochastic gradient descent. Our network model achieves 95.0% accuracy with 60000 training images. The inset shows that with FFI, the learning is faster at the beginning, compared to sparse coding.

Figure 5.3 presents the testing accuracy as the training proceeds. After every 50 batches, we simulate the network with testing images, without change the synaptic weights, to get the testing accuracy. The network is trained with random initialization several times and the performance is very stable. On average, it achieves 94.2% accuracy with 20000 training images, and achieves 95.0% accuracy with 60000 training images. Our network is also able to perform unsupervised training with the absence of reward signal. The performance degradation is very small (0.1%), indicating that the reward signal is not critical for this relatively simple task. The result is compared with the sparse coding algorithm (Lee et al. 2006), which uses stochastic gradient method in training. Sparse coding is an unsupervised machine learning algorithm that learns both the weights (i.e., dictionary) and the sparse representations of the data. Since the data representation in our network is equivalent to only 6 binary bits, 6-bit data precision is used in the sparse coding algorithm for a fair comparison (Chen et al. 2015). The precision of weights is 32-bit in both our algorithm and sparse coding. The results show that sparse coding algorithm performs better than our network, because it uses gradient based optimization method, which is not biologically plausible. It is worth mentioning that when the number of training image was very limited, like about 500, the network still had above 80% accuracy, which was much better than sparse coding. The ability of fast learning in such a small number of trials is observed in live animals as well, which is critical to survival and evolution. It has enabled the nervous systems to tend to learn faster than the machine learning algorithms that are currently available. Querlioz (2013, 12:288-295) proposed a lateral inhibition spiking neural network trained with STDP. They need 6400 excitatory neurons to achieve 95.0% accuracy on the same dataset. The FFI network is 3X more efficient than the lateral inhibition network.

5.3.2. Important Factors to Train the Network

Several experiments are conducted to investigate the important factors in the training step. The first one is the randomization in the initialization stage. A good random initialization is crucial to high learning performance. There are two aspects of initialization: connectivity and weights. The connectivity is defined as the percentage of input neurons that each output neuron (e.g. E or I) receives innervation from. Figure 5.4 shows that the connectivity needs to be larger than a minimum value (13%) to achieve a good training result. In addition, when the network connectivity is larger than 80%, the learning accuracy starts to degrade slightly. Thus, in this paper each E or I neuron receives innervation from random 50% of the input neurons.

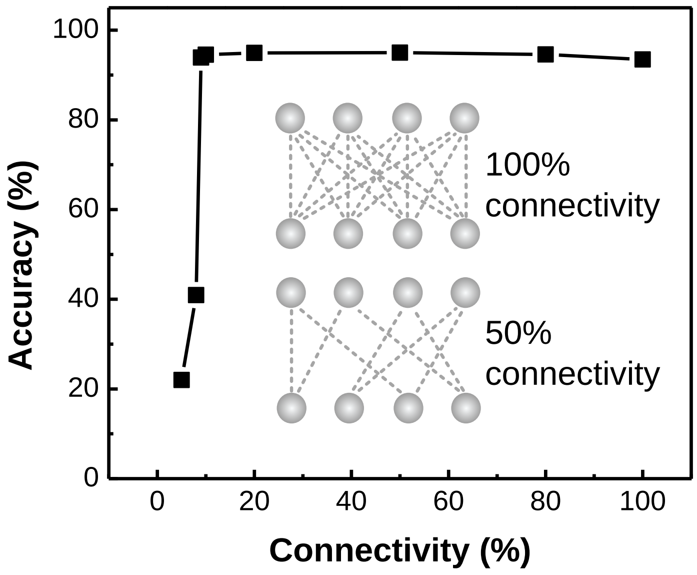


Figure 5.4. The effect of connectivity: The prediction accuracy of the FFI networks with various levels of connectivity, which is defined as the percentage of input neurons that each output neuron (e.g. E or I) receives innervation from.

Each synaptic connection is also initialized by a random weight. It is known that, if all E and I neurons are connected to all the input neurons and all the synapses have the same weight, no learning can happen, because all the neurons are identical and all the weight changes are identical too. The random initialization of the weights is needed to break the symmetry. Figure 5.5 shows the learning curve of the network with different random weight initialization when all E and I neurons are connected to all the input neurons. The randomness ranges from 100% to 10%. 100% means the weights are initialized with random numbers between 0 and 1. 10% means the weights are initialized with random numbers between 0.45 and 0.55. All the random numbers are drawn from uniform distribution. The figure indicates that enough randomness is needed for a good training result. The learning performance is better with more randomness. When the level of randomness is low, the training of weights may be stuck at non-optimal values and is more difficult to converge. Therefore, the level of randomness needs to be high enough to perform statistical training.

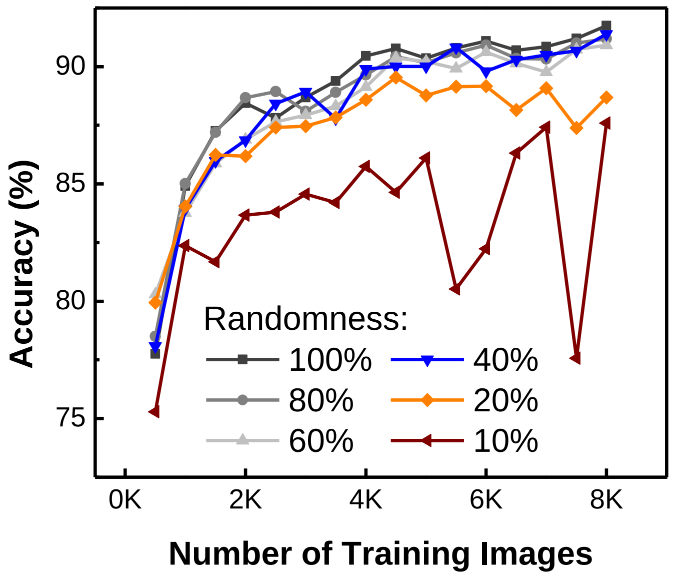


Figure 5.5. The effect of weight randomization: The training curve of networks with 50% connectivity but different initial ranges of random weights. The randomness ranges from 100% to 10%. 100% means the weights are initialized with random numbers between 0 and 1. 10% means the weights are initialized with random numbers between 0.45 and 0.55. All the random numbers are drawn from uniform distribution. The figure shows that if initial randomness is too small (<10%), the training cannot converge.

When the E and I neurons receive innervation only from part of the input neurons randomly (i.e., connectivity less than 100%), more randomness is added to the initialization and better accuracy is achieved. Other experiments show that 50% connectivity is an optimized initialization for this dataset. And at least 10% connectivity is needed to capture meaningful input patterns and have good learning performance. Another reason why random initialization is so important to the statistical learning is because Hebbian learning is a process of positive feedback. Useful differentiation of features will be strengthened during training. With more variations in the system, the mechanism of positive feedback will learn how to amplify and separate the features for various classes. On the other side, if the input variation is too low that even after the amplification by the feedback, there is not enough separation to be created, then this non-linear training procedure will fail; as a result, we observe the sudden drops in accuracy.

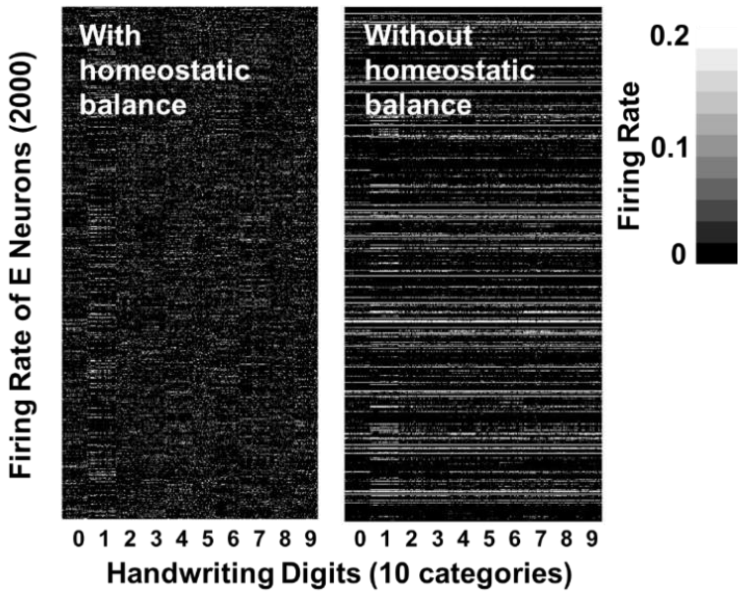


Figure 5.6. The effect of homeostatic balance: The firing rate of E neurons for different classes of images, after training with and without homeostatic balance. Without homeostatic balance, some neurons may be stuck at the constantly firing state (white) or the non-firing state (dark), failing to differentiate the classes with different firing patterns.

As Hebbian learning is a process of positive feedback, it is very important to keep the homeostatic balance. As shown in Figure 5.6, without homeostatic balance, the neurons will be stuck at constantly firing state or non-firing state during training. These neurons cannot extract any useful feature and are useless to the learning. It needs to keep the long-term firing rate of all neurons at a similar value by the balancing methods. Thus, each neuron will only respond when particular input features arise.

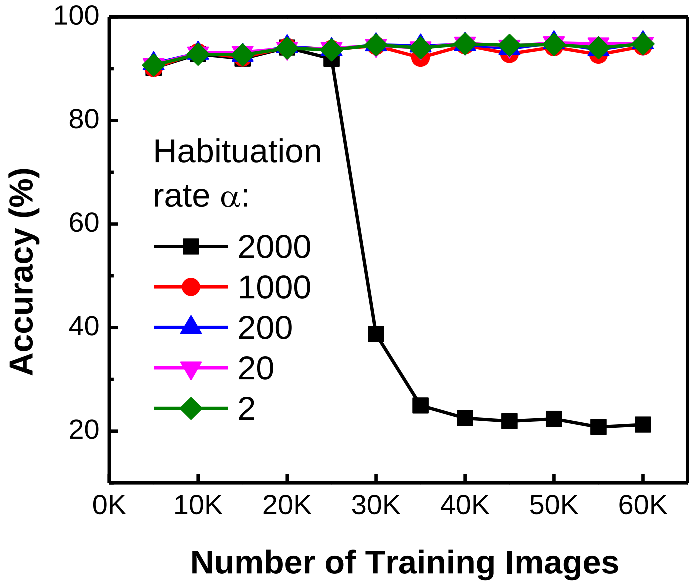


Figure 5.7. The effect of habituation rate: The learning curve of networks with different habituation rates α in Equation (18). If the learning rate doesn't decay or decays too slowly, the training cannot complete.

For the synapse, it is also important to make sure the learning converges, i.e., the synaptic weight should be stable toward the end of training. In our algorithm, convergence is guaranteed by reducing the learning rate during training. It is inspired by the habituation of the synaptic plasticity in biological nervous systems. The more times a synaptic weight is changed, the less it can be changed by the same amount of stimulus. In our network, the learning rate of each synapse is reversed proportional to the number of times it has been updated. The speed of habituation determines the speed of convergence. As shown in

Figure 5.7, if the habituation is too slow, the training will fail at some point. The speed of habituation needs to be optimized for high accuracy.

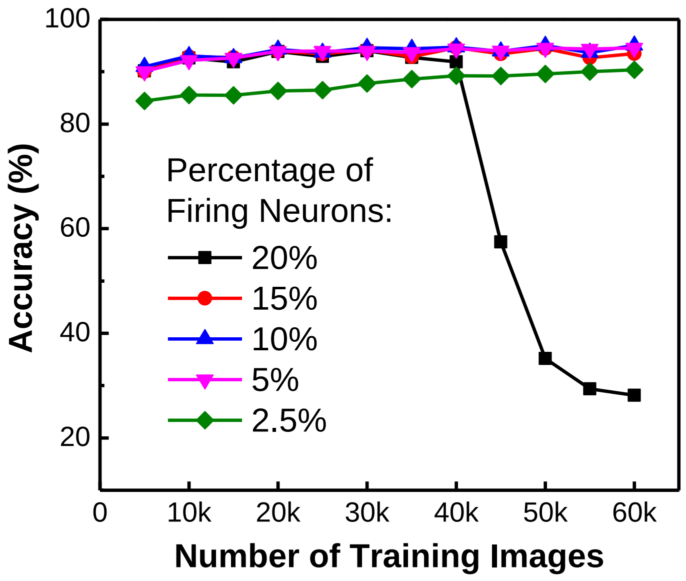


Figure 5.8. The effect of sparsity: The training curves with different sparsity, which is defined as the percentage of firing neurons in the E layer. If too many neurons are firing together, the training may fail due to the incapability to differentiate various classes; if too few neurons are firing, the accuracy degrades too. The optimal number is around 10% because there are totally 10 classes in this learning task.

For the neurons, the parameters that impact learning include the threshold of neurons and the sparsity. The sparsity is controlled by the threshold θ. Higher threshold results in fewer number of firing neurons and lower threshold results in more number of firing neurons. From the learning aspect, it prefers less overlaps that a neuron fires across different class of images, in order to better differentiate one class from another. In addition, it prefers all output neurons can be utilized to extract features for the entire dataset, i.e., the full feature space is utilized. Figure 5.8 shows that the average percentage of firing neurons for one image affects the accuracy. For this dataset, since there are 10 different classes, when the percentage of firing neurons is about 10%, it achieves the best accuracy. Figure 5.8 further indicates that that if the number of firing neurons is too large, the training will fail at some point due to too many overlaps in the feature space.

5.3.3. Role of Feedforward Inhibition in Learning

The neural network model can help us understand the role of the FFI motif in learning, such as the benefit of adding this small group of interneurons. From both biological experiments and previous neural networks, we know that FF excitatory neurons only are sufficient to extract sparse features and classify objects. Yet biological studies in vivo further indicated that the suppression of interneurons degraded the learning process, implying that critical role of the FFI motif. To investigate the function of FFI, we conducted a comparative study by training two similar neural networks, one with inhibition (Model A) and the other without inhibition (Model B). The rest of the network model, such as the number of neurons, connectivity, etc., are all identical. The same rewarded SRDP rule was applied to both models. For a fair comparison, the threshold of E neurons in Model B was increased to obtain a similar group firing rate as that in Model A.

The first experiment studied the dependence of learning efficacy on the number of excitatory neurons, while the number of interneurons was kept constant. Figure 5.9 presents the result. As the number of E neurons increases, the accuracy is improved. The accuracy is more sensitive to the size of E, when the number of E neurons is low; when the size of E keeps increasing, the accuracy increased more slowly and eventually became saturated. Moreover, the results in Figure 5.9 illustrate that at the same number of E neurons, Model A, the network with FFI, achieves higher accuracy than Model B. To achieve the same accuracy of 95.0%, Model A only needs 2000 excitatory neurons with 100 interneurons while Model B, the model without the FFI motif, required 5200 neurons. Thus, adding a small inhibitory layer can not only improve the final accuracy, but also significantly reduce the number of E neurons needed to achieve a specific accuracy (i.e., in this case, a 2.6X reduction), resulting in much greater hardware efficiency. The improved hardware compactness will in turn bring better energy efficiency and computing speed. Furthermore, we trained Model A with various numbers of I neurons. The result showed that as long as the number of I neurons is larger than 10, the network achieved similar accuracy.

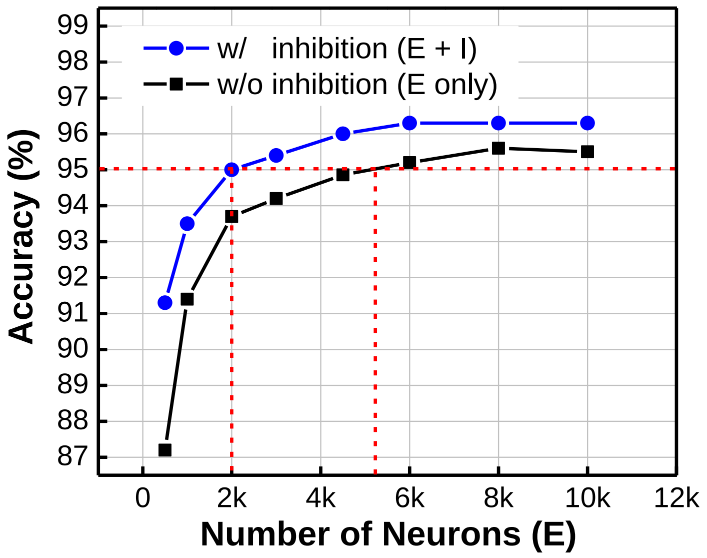


Figure 5.9. The prediction accuracy as a function of the number of E neurons in the network model, with or without the inhibition layer (100 I neurons). To achieve 95.0% accuracy, the network with feedforward inhibition only needs 2000 E neurons, while the network without inhibition needs 5200 E neurons. The network with inhibition shows both higher accuracy and higher hardware efficiency (i.e., 2.6X reduction in the number of excitation neurons at 95.0%).

More experiments are performed on Model A, the FFI motif with 2000 E neurons and 100 I neurons to study the role of inhibition. If the interneurons are removed from the network after training, the excitatory neurons started to fire at a much higher rate, and had a lot of overlaps across different classes, which was consistent with the observations from biological experiments on the locust olfactory system (Perez-Orive et al. 2004, 24:6037-6047). In the biological experiments, blockade of inhibition in the mushroom body caused neurons to become responsive to a much larger number of odors and lose the ability to discriminate between them. Similarly, the prediction accuracy of our network dropped from 95.0% to 47.7% after removal of interneurons due to too many firing neurons, which led to less selective feature extraction. Both the network model and the biological experiment demonstrated similar roles of inhibition in the regulation of firing rate and the maintenance of sparsity in the response. To further prove this, we increased the threshold of E neurons for the network after removing I neurons, so that the firing rate of E neurons was restored to that with I neurons. With this change, the prediction accuracy was improved back to 92.5%, but still significantly lower than that from Model A at 95.0%. Based on these experiments, we conclude that the first role of the FFI motif is to regulate the firing rate and the sparsity of E neurons.

To better understand the interaction between I and E neurons, the plasticity of the inhibitory synapses from I to E are turned off. The network still has 2000 E neurons and 100 I neurons. When the inhibitory synapses were fixed at a constant weight, each E neurons received the same amount of inhibition proportional to the group firing rate of I neurons. This network achieved 94.7% accuracy after training. Note that Model B with 2000 E neurons, but without inhibition, achieved 93.7% accuracy. Comparing these two results, the 1% improvement in accuracy is because the I neurons can dynamically regulate the E neurons based on the input image, while without inhibition the regulation of E neurons is by a fixed threshold regardless of the input. This hypothesis is also proved by Figure 5.10 and Figure 5.11. Figure 5.10 shows less batch to batch variation of the average percentage of firing neurons for the network with inhibition. Figure 5.11 shows that the average firing rate of E neurons without inhibition and the average firing rate of I neurons are strongly correlated. The training of the network can strengthen such correlation. The correlation coefficient between E firing and I firing is increased from 0.46 (before training) to 0.74 (after training). Therefore, we conclude that the second role of the FFI motif is to provide the sensitivity to each input during the regulation of E neurons, beyond the indirect and delayed FB inhibition from E neurons.

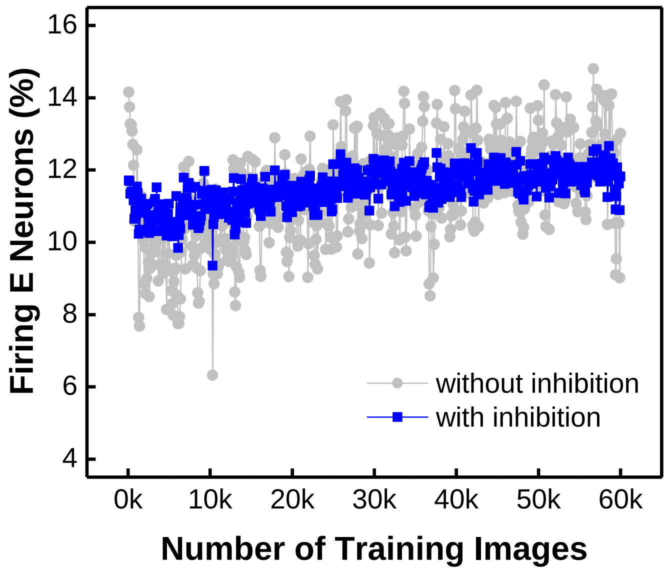
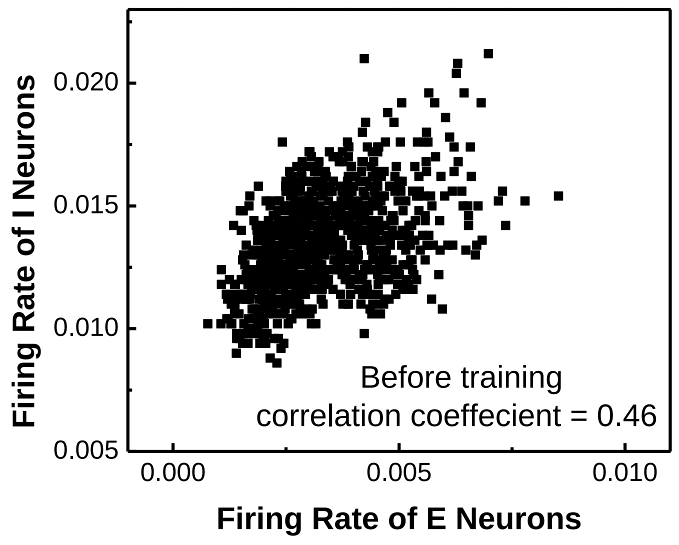


Figure 5.10. Thresholding: The percentage of firing E neurons (i.e., sparsity) during model training, with and without inhibition. Each data point represents the average percentage of firing E neurons for a batch of images (100 images). The network with feedforward inhibition has much smaller batch to batch variations. Thus, FFI helps regulate the firing rate of E neurons.



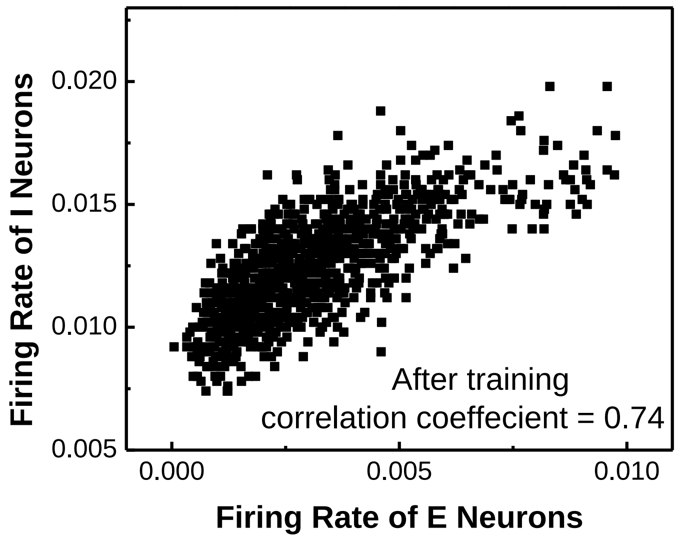


Figure 5.11. Dynamic thresholding: The figures show the correlation between the average firing rate of E neurons without inhibition and the average firing rate of I neurons before and after training. There are 1000 data points in each figure for 1000 randomly selected images. Each data point represents the average firing rate of E neurons without inhibition (x axis) and I neurons (y axis). The correlation coefficient is 0.46 before training, while the training process strengthens the correlation to be 0.74. This change indicates that FFI responds to the same input as E and carries correlated information to suppress E, beyond simple thresholding.

Besides the interaction between E and I, the I neurons are also stimulated by the same input as that to E. The only difference between E and I is their size: there are much more E neurons than I neurons. Therefore, some fundamental questions in computing are what the interneurons learn from the input, and how the learning by I help improve the accuracy. To understand that, we reproduced the classifier with 10 output neurons and added that after the inhibition layer. These 10 neurons received input from the interneurons and predicted the digits. After training this new classifier, the result showed that the interneurons achieve 68% accuracy with only 100 neurons. A more careful examination revealed that the images that can be successfully recognized by the interneurons are more standard (i.e., closer to the average image in each category) than those unrecognized, as shown in Figure 5.12. Here, how standard an image looks is measured by the Euclidian distance from the image to its corresponding cluster center assigned by K-means, which is a representative unsupervised clustering algorithm. Images with longer distance from their center are generally more difficult to be recognized because they contain more unusual features. For the examples in Figure 5.12, the correct group can represent majority of the images in the dataset. Their patterns have more common features. The interneurons can make good prediction on these images. On the contrast, the wrong group has more uncommon features which don’t appear often in the dataset, i.e., statistically they are farther away from the cluster center. Thus, the interneurons cannot recognize them well. Figure 5.12 shows the statistical accuracy with respect to this distance for three different motifs, the inhibitory layer, the excitatory layer without inhibition (Model B) and the excitatory layer with inhibition (Model A). They show the same trend but the excitatory neurons have much higher accuracy than the interneurons, which can be attributed to the difference in their size. The difference between Model A and Model B is from the images near the centers. When the distance is smaller than 0.04, the interneurons have accuracy above 80%, which boosts the accuracy of excitatory neurons from 90% to nearly 100%. Because the images with small distance are the majority part of the data, this improvement is important to increase the overall accuracy.

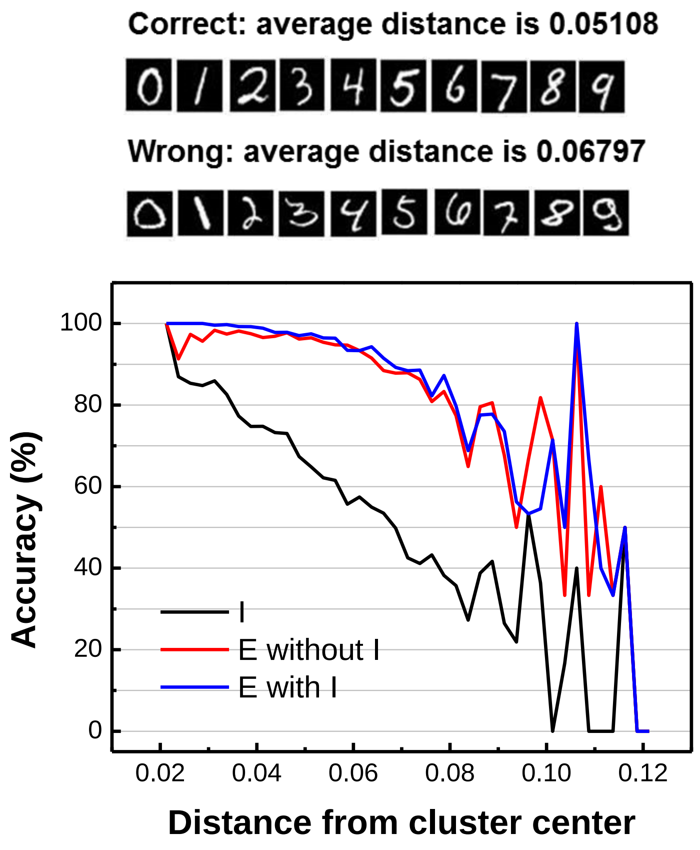
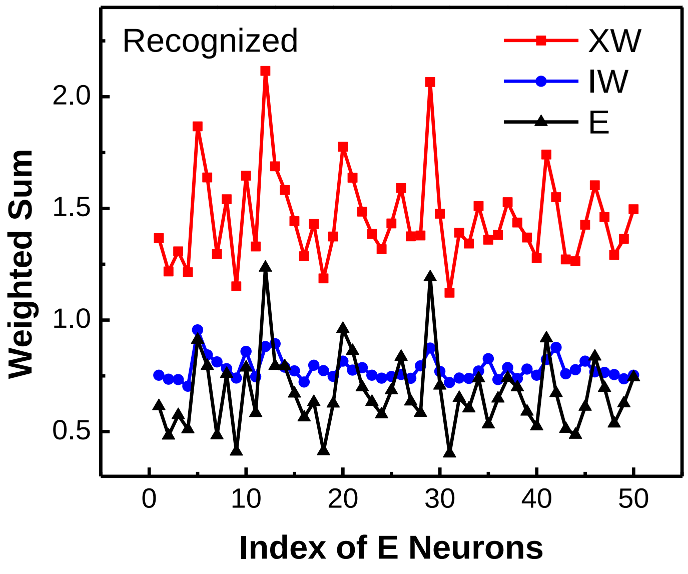


Figure 5.12. The classification results of the inhibition layer. The first figure shows the example images that I neurons can and cannot recognize after training. The second figure shows the prediction accuracy as a function of the distance from an image to its cluster center, using the K-means method.



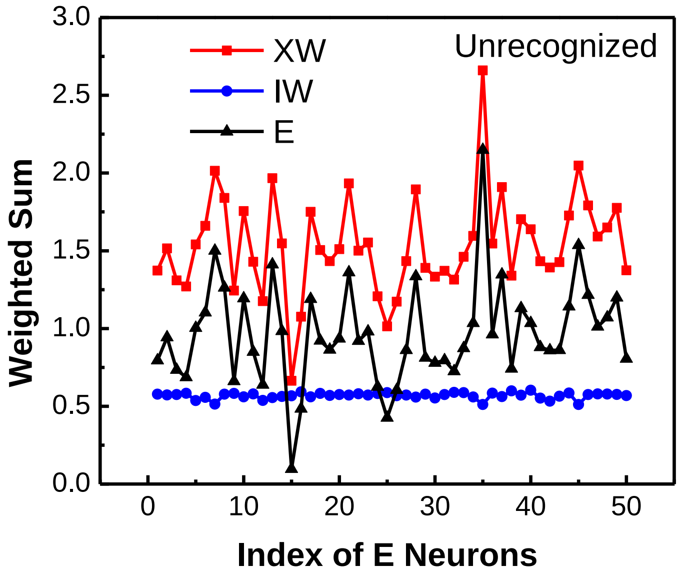


Figure 5.13. The total weighed sum of input from excitatory path and inhibitory path of 50 random selected E neurons. The left figure is the image can be recognized and the right figure is the image cannot be recognized.

Figure 5.13 shows the total current (i.e. weighed sum of input) from excitatory path and inhibitory path of 50 E neurons for two different input images. For the image that is well recognized by the network, the excitatory current and the inhibitory current have strong correlation. For the image that is not well recognized by the network, the inhibitory current has no correlation with the excitatory current. In conclusion, the third role of inhibition is the FFI motif is to recognize a major portion of the images with a concise layer, and increase the accuracy of the excitatory neurons on these images.

5.3.4. Comparison with Relate Works

As a summary, a comparison of different algorithms on MNIST dataset is shown in Table 9. Compared to biologically plausible methods, our method achieves higher accuracy with less number of neurons; compared to artificial learning algorithms that have better accuracy, our method only uses feedforward computation, without resorting to expensive backpropagation on training, and thus, enhances computation efficiency.

Table 9. Comparison between the neural network models in this paper and related works.

Model	Data representation	Learning rules	Number of neurons	Number of parameters	Number of images	Accuracy
Insect mushroom body model (Huerta 2009)	Spike	Rewarded STDP	50000	5E5	60000	87%
Two-layer SNN (Querlioz 2013)	Spike	STDP	300	2.4E5	60000x3	93.5%
Lateral inhibitory SNN (Diehl 2015)	Spike	STDP	6400	4.6E7	200000	95.0%
This work (w/ FFI)	Spike rate	Rewarded SRDP	2100	8.4E5	60000	95.0%
This work (w/o FFI)	Spike rate	Rewarded SRDP	6000	2.4E6	60000	95.2%
Spiking RBM (Neftci 2013)	Spike rate	Contrastive divergence	500	3.9E5	20000	92.6%
Sparse Coding (Chen 2015)	6-bit number	Gradient descent	300	3E4	60000x10	95.9%
Two-layer NN (LeCun 1998)	Floating-point number	Gradient descent	1000	7.8E5	60000	95.5%
Spiking CNN (Panda 2016)	Spike timing	Regenerative learning	5.6E4	1.2E5	60000	99.08%

6. Future Work

There are two directions for the future work. On the hardware side, the proposed algorithm can be implemented on a chip using either SRAM or RRAM array as the synapses. The computing, programming and controlling circuits need to be designed with the target of high performance and low power. The early stage implementation with FPGA can also be performed as a proof of concept. One major concern for the hardware implementation is the data and parameter precision. It is very important to use minimum number of bit which can keep good accuracy, in order to save the hardware resources, and reduce latency and power.

For the bio-inspired algorithm, it is very interesting to try more challenging dataset such as CIFAR10. Proper tuning of parameters and even the learning rules are important to make the algorithm work well. It is also interesting to explore deeper neural networks with this algorithm since several more layers of neural networks are observed in the brain. In addition, this algorithm has the potential to realize online learning, which is able to learn from data continuously as the data distribution changes over time.

7. Conclusion

This dissertation aims at the potential of machine learning on-a-chip. For this purpose, state-of-the-art device options are firstly reviewed. For memory device, emerging non-volatile memories are very promising synaptic devices to enable large scale parallel computing in machine learning. Thus, PRAM and STT-MRAM are extensively studied and modeled within the proposed hierarchical framework. In the design perspective, a new performance metric, State Transition Curve, is proposed for the assessment of PRAM cell and to provide valuable design insights. In addition, various simulations are conducted to investigate the performance, optimization, variability, reliability, and scalability of these two memories. As a hardware implementation practice, peripheral programming circuitry is designed for the parallel programming of RRAM cross-point array as the synapses in neural networks. The simulation shows 900X improvement in speed of dictionary learning. On the algorithm side, a bio-plausible feedforward inhibition motif is developed with leaky integrate-and-fire neurons and SRDP Hebbian learning rule. It shows great performance and high efficiency of both computation and hardware, achieving 95% testing accuracy on MNIST dataset with 30X less number of computations than sparse coding. And the feedforward inhibition is shown to save 3X on hardware resources. In addition, the reason why feedforward inhibition can improve the hardware efficiency is thoroughly studied.

REFERENCES

Alon, Uri. 2007. Network motifs: theory and experimental approaches. *Nature Reviews Genetics* 8(6):450-461.

Assisi, Collins, Mark Stopfer, Gilles Laurent, and Maxim Bazhenov. 2007. Adaptive regulation of sparseness by feedforward inhibition. *Nature neuroscience* 10(9):1176.

Bahrami, Shahram, and Finn Drabløs. 2016. Gene regulation in the immediate-early response process. *Advances in biological regulation* 62:37-49.

Baig, Mirza M., Mian M. Awais, and El-Sayed M. El-Alfy. 2017. AdaBoost-based artificial neural network learning. *Neurocomputing* 248:120-126.

Baldassi, Carlo. 2009. Generalization learning in a perceptron with binary synapses." *Journal of Statistical Physics* 136(5): 902-916.

Baldassi, Carlo, Federica Gerace, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. 2016. Learning may need only a few bits of synaptic precision. *Physical Review* 93(5):052313.

Bedeschi, Ferdinando, Rich Fackenthal, Claudio Resta, Enzo Michele Donze, Meenatchi Jagasivamani, Egidio Cassiodoro Buda, Fabio Pellizzer et al. 2009. A bipolar-selected phase change memory featuring multi-level cell storage. *IEEE Journal of Solid-State Circuits* 44(1):217-227.

Bi, Guo-qiang, and Mu-ming Poo. 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of neuroscience* 18(24):10464-10472.

Bousquet, Olivier, and Léon Bottou. 2008. The tradeoffs of large scale learning. In *Advances in neural information processing systems* 161-168.

Burr, Geoffrey W., Matthew J. Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi et al. 2010. Phase change memory technology. *Journal of Vacuum Science & Technology B, Nanotechnology and Microelectronics: Materials, Processing, Measurement, and Phenomena* 28(2):223-262.

Cao, Yongqiang, Yang Chen, and Deepak Khosla. 2015. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision* 113(1):54-66.

Chen, Pai-Yu, Binbin Lin, I-Ting Wang, Tuo-Hung Hou, Jieping Ye, Sarma Vrudhula, Jae-sun Seo, Yu Cao, and Shimeng Yu. 2015. Mitigating effects of non-ideal synaptic device characteristics for on-chip learning. *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on* 194-199.

Chen, Tianshi, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49(4):269-284.

Chen, Yunji, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM* 59(11):105-112.

Chen, Yu-Hsin, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52(1):127-138.

Chi, Ping, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *Proceedings of the 43rd International Symposium on Computer Architecture*, 27-39.

Chun, Ki Chul, Hui Zhao, Jonathan D. Harms, Tae-Hyoung Kim, Jian-Ping Wang, and Chris H. Kim. 2013. A scaling roadmap and performance evaluation of in-plane and perpendicular MTJ based STT-MRAMs for high-density cache memory. *IEEE Journal of Solid-State Circuits* 48(2):598-610.

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv* preprint arXiv:1412.7024.

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems* 3123-3131.

Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv* preprint arXiv:1602.02830.

Diao, Zhitao, Zhanjie Li, Shengyuang Wang, Yunfei Ding, Alex Panchula, Eugene Chen, Lien-Chang Wang, and Yiming Huai. 2007. Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory. *Journal of Physics: Condensed Matter* 19(16):165209.

Diehl, Peter U., and Matthew Cook. 2015. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience* 9.

Diehl, Peter U., Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. *International Joint Conference on Neural Networks* 1-8.

Du, Zidong, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. *ACM SIGARCH Computer Architecture News* 43(3):92-104.

Ernst, K. D., J. Boeckh, and V. Boeckh. 1977. A neuroanatomical study on the organization of the central antennal pathways in insects. *Cell and tissue research* 176(3):285-308.

Faber, Louis-Barthelemy, Weisheng Zhao, Jacques-Oliver Klein, Thibaut Devolder, and Claude Chappert. 2009. Dynamic compact model of spin-transfer torque based magnetic tunnel junction (MTJ). *Design & Technology of Integrated Systems in Nanoscal Era, 2009. DTIS'09. 4th International Conference on* 130-135.

Farabet, Clément, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. Neuflow: A runtime reconfigurable dataflow processor for vision. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, 109-116.

Fraser, Nicholas J., Yaman Umuroglu, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Scaling binarized neural networks on reconfigurable logic. *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms* 25-30.

Furber, Steve. 2016. Large-scale neuromorphic computing systems. *Journal of neural engineering* 13(5):051001.

Galizia, C. Giovanni, and Silke Sachse. 2010. Odor coding in insects. *The neurobiology of olfaction* 35-70.

Garbin, Daniele, Olivier Bichler, Elisa Vianello, Quentin Rafhay, Christian Gamrat, L. Perniola, G. Ghibaudo, and B. DeSalvo. 2014. Variability-tolerant convolutional neural network for pattern recognition applications based on OxRAM synapses. *Electron Devices Meeting (IEDM), 2014 IEEE International*, 28-4.

Gokhale, Vinayak, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. *IEEE Conference on Computer Vision and Pattern Recognition Workshops* 682-687.

Gokmen, Tayfun, and Yurii Vlasov. 2016. Acceleration of deep neural network training with resistive cross-point devices: design considerations. *Frontiers in neuroscience* 10.

Guan, Ximeng, Shimeng Yu, and H-S. Philip Wong. 2012. A SPICE compact model of metal oxide resistive switching memory with variations. *IEEE electron device letters* 33(10):1405-1407.

Gupta, Suyog, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. *International Conference on Machine Learning* 37:1737-1746.

Gysel, Philipp, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented approximation of convolutional neural networks. *arXiv* preprint arXiv:1604.03168.

Haider, Bilal, Alvaro Duque, Andrea R. Hasenstaub, and David A. McCormick. 2006. Neocortical network activity in vivo is generated through a dynamic balance of excitation and inhibition. *Journal of Neuroscience* 26(17):4535-4545.

Han, Song, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv* preprint arXiv:1510.00149.

Han, Song, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: efficient inference engine on compressed deep neural network. *International Symposium on Computer Architecture* 243-254.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition* 770-778.

Hollis, Paul W., John S. Harper, and John J. Paulos. 1990. The effects of precision constraints in a backpropagation learning network. *Neural Computation* 2(3):363-373.

Hong, Chaoqun, Jun Yu, Jian Wan, Dacheng Tao, and Meng Wang. 2015. Multimodal deep autoencoder for human pose recovery. *IEEE Transactions on Image Processing* 24(12):5659-5670.

Huerta, Ramón, and Thomas Nowotny. 2009. Fast and robust learning by reinforcement signals: explorations in the insect brain. *Neural computation* 21(8):2123-2151.

Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv* preprint arXiv:1602.07360.

ILSVRC. “ImageNet Large Scale Visual Recognition Challenge 2017.” http://image-net.org/challenges/LSVRC/2017/results.

Itri, A., D. Ielmini, A. L. Lacaitat, A. Pirovano, E. Pellizzer, and R. Bez. 2004. Analysis of phase-transformation dynamics and estimation of amorphous-chalcogenide fraction in phase-change memories. *Reliability Physics Symposium Proceedings, 2004. 42nd Annual. 2004 IEEE International* 209-215.

Jo, Sung Hyun, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. 2010. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters* 10(4):1297-1301.

Johnson, William A. 1939. Reaction kinetics in process of nucleation and growth. *Transaction of AIME* 135:416-458.

Jouppi, Norman P., Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates et al. 2017. In-datacenter performance analysis of a tensor processing unit. *arXiv* preprint arXiv:1704.04760.

Kammerer, Jean-Baptiste, Morgan Madec, and Luc Hébrard. 2010. Compact modeling of a magnetic tunnel junction—Part I: Dynamic magnetization model. *IEEE Transactions on Electron Devices* 57(6):1408-1415.

Karpathy, Andrej. “What I Learned from Competing Against a ConvNet on ImageNet.” Andrej Karpathy blog. http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/.

Kelsom, Corey, and Wange Lu. 2013. Development and specification of GABAergic cortical interneurons. *Cell & bioscience* 3(1):19.

Kim, Kinarn, and Su Jin Ahn. 2005. Reliability investigations for manufacturable high density PRAM. *Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International* 157-162.

Kim, Minje, and Paris Smaragdis. 2016. Bitwise neural networks. *arXiv* preprint arXiv:1601.06071.

King, Paul D., Joel Zylberberg, and Michael R. DeWeese. 2013. Inhibitory interneurons decorrelate excitatory cells to drive sparse code formation in a spiking model of V1. *Journal of Neuroscience* 33(13):5475-5485.

Krizhevsky, Alex, and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 1097-1105.

Kwong, K. C., Lin Li, Jin He, and Mansun Chan. 2008. Verilog-A model for phase change memory simulation. *Solid-State and Integrated-Circuit Technology, 2008. ICSICT 2008. 9th International Conference on* 492-495.

Lacaita, A. L., A. Redaelli, D. Ielmini, F. Pellizzer, A. Pirovano, A. Benvenuti, and R. Bez. 2004. Electrothermal and phase-change dynamics in chalcogenide-based memories. *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International* 911-914.

Lane, Nicholas D., and Petko Georgiev. 2015. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* 117-122.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278-2324.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521:436-444.

Lee, Honglak, Alexis Battle, Rajat Raina, and Andrew Y. Ng. 2007. Efficient sparse coding algorithms. *Advances in neural information processing systems* 801-808.

Le Masson, Gwendal, Sylvie Renaud-Le Masson, Damien Debay, and Thierry Bal. 2002. Feedback inhibition controls spike transfer in hybrid thalamic circuits. *Nature* 417(6891):854.

Li, Fengfu, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv* preprint arXiv:1605.04711.

Li, Lin, and Mansun Chan. 2008. Scaling analysis of phase change memory (PCM) driving devices. *Electron Devices and Solid-State Circuits, 2008. EDSSC 2008. IEEE International Conference on* 1-4.

Li, Yiming, Shao-Ming Yu, Chih-Hong Hwang, and Yi-Ting Kuo. 2008. Temperature dependence on the contact size of GeSbTe films for phase change memories. *Journal of Computational Electronics* 7(3):138-141.

Lin, C. J., S. H. Kang, Y. J. Wang, K. Lee, X. Zhu, W. C. Chen, X. Li et al. 2009. 45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell." In *Electron Devices Meeting (IEDM), 2009 IEEE International* 1-4.

Lin, Darryl D., and Sachin S. Talathi. 2016. Overcoming challenges in fixed point training of deep convolutional networks. *arXiv* preprint arXiv:1607.02241.

Liu, Weibo, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. 2017. A survey of deep neural network architectures and their applications. *Neurocomputing* 234:11-26.

Lu, H. M., W. T. Zheng, and Q. Jiang. 2007. Saturation magnetization of ferromagnetic and ferrimagnetic nanocrystals at room temperature. *Journal of Physics D: Applied Physics* 40(2):320.

Luo, Tao, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen. 2017. DaDianNao: a neural network supercomputer. *IEEE Transactions on Computers* 66(1):73-88.

Ma, Yufei, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. 2016. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. *Field Programmable Logic and Applications (FPL), 2016 26th International Conference on* 1-8.

Madec, Morgan, Jean-Baptiste Kammerer, and Luc Hébrard. 2010. Compact modeling of a magnetic tunnel junction—Part II: Tunneling current model. *IEEE Transactions on Electron Devices* 57(6):1416-1424.

Merolla, Paul A., John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson et al. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345(6197):668-673.

Merolla, Paul, Rathinakumar Appuswamy, John V. Arthur, Steven K. Esser, and Dharmendra S. Modha. 2016. Deep neural networks are robust to weight binarization and other non-linear distortions. *CoRR* abs/1606.01981.

Misra, Janardan, and Indranil Saha. 2016. Artificial neural networks in hardware: a survey of two decades of progress. *Neurocomputing* 74(1-3):239-255.

Neftci, Emre, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. 2013. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in neuroscience* 7.

Nurvitadhi, Eriko, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC." *Field-Programmable Technology (FPT), 2016 International Conference on* 77-84.

Pan, Xipeng, Lingqiao Li, Huihua Yang, Zhenbing Liu, Jinxin Yang, Lingling Zhao, and Yongxian Fan. 2017. Accurate segmentation of nuclei in pathological images via sparse reconstruction and deep convolutional networks. *Neurocomputing* 229:88-99.

Panda, Priyadarshini, and Kaushik Roy. 2016. Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. *Neural Networks (IJCNN), 2016 International Joint Conference on* 299-306.

Perez-Orive, Javier, Maxim Bazhenov, and Gilles Laurent. 2004. Intrinsic and circuit properties favor coincidence detection for decoding oscillatory input. *Journal of Neuroscience* 24(26):6037-6047.

Plagianakos, V. P., and M. N. Vrahatis. 1999. Training neural networks with 3-bit integer weights. *Genetic and Evolutionary Computation Conference* 910-915.

Pouille, Frédéric, Antonia Marin-Burgin, Hillel Adesnik, Bassam V. Atallah, and Massimo Scanziani. 2009. Input normalization by global feedforward inhibition expands cortical dynamic range." *Nature neuroscience* 12(12):1577-1585.

Prezioso, Mirko, Farnood Merrikh-Bayat, B. D. Hoskins, G. C. Adam, Konstantin K. Likharev, and Dmitri B. Strukov. 2015. Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521(7550):61-64.

Qiu, Jiantao, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* 26-35.

Querlioz, Damien, Olivier Bichler, Philippe Dollfus, and Christian Gamrat. 2013. Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology* 12(3):288-295.

Ralph, Daniel C., and Mark D. Stiles. 2008. Spin transfer torques. *Journal of Magnetism and Magnetic Materials* 320(7):1190-1216.

Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision* 525-542.

Sarpeshkar, Rahul. 1998. Analog versus digital: extrapolating from electronics to neurobiology. *Neural computation* 10(7):1601-1638.

Scardapane, Simone, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. 2017. Group sparse regularization for deep neural networks. *Neurocomputing* 241:81-89.

Schmidhuber, Jürgen. 2015. Deep learning in neural networks: An overview. *Neural networks* 61:85-117.

Seo, Jae-sun, Bernard Brezzo, Yong Liu, Benjamin D. Parker, Steven K. Esser, Robert K. Montoye, Bipin Rajendran et al. 2011. A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. *Custom Integrated Circuits Conference (CICC), 2011 IEEE* 1-4.

Sharad, Mrigank, Charles Augustine, Georgios Panagopoulos, and Kaushik Roy. 2012. Spin-based neuron model with domain-wall magnets as synapse. *IEEE Transactions on Nanotechnology* 11(4):843-853.

Sillito, A. M. 1977. Inhibitory processes underlying the directional specificity of simple, complex and hypercomplex cells in the cat's visual cortex. *The Journal of Physiology* 271(3):699-720.

Simonyan, Karen, and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv* preprint arXiv:1409.1556.

Skorheim, Steven, Peter Lonjers, and Maxim Bazhenov. 2014. A spiking network model of decision making employing rewarded STDP. *PloS one* 9(3):e90821.

Song, Sen, Kenneth D. Miller, and Larry F. Abbott. 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience* 3(9):919-926.

Sporns, Olaf, and Rolf Kötter. 2004. Motifs in brain networks. *PLoS biology* 2(11):e369.

Stopfer, Mark, Vivek Jayaraman, and Gilles Laurent. 2003. Intensity versus identity coding in an olfactory system. *Neuron* 39(6):991-1004.

Stromatias, Evangelos, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B. Furber, and Shih-Chii Liu. 2015. Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience* 9.

Sun, Kai, Jiangshe Zhang, Chunxia Zhang, and Junying Hu. 2017. Generalized extreme learning machine autoencoder and a new deep neural network." *Neurocomputing* 230:374-381.

Sawaguchi, T. O. S. H. I. Y. U. K. I., I. T. A. R. U. Yamane, and K. I. S. O. U. Kubota. 1996. Application of the GABA antagonist bicuculline to the premotor cortex reduces the ability to withhold reaching movements by well-trained monkeys in visually guided reaching task. *Journal of Neurophysiology* 75(5):2150-2156.

Szlam, Arthur D., Karol Gregor, and Yann L. Cun. 2011. Structured sparse coding via lateral inhibition. *Advances in Neural Information Processing Systems* 1116-1124.

Tosic, Ivana, and Pascal Frossard. 2011. Dictionary learning. *IEEE Signal Processing Magazine* 28(2):27-38.

Tsumoto, T., W. Eckart, and O. D. Creutzfeldt. 1979. Modification of orientation sensitivity of cat visual cortex neurons by removal of GABA-mediated inhibition. *Experimental Brain Research* 34(2):351-363.

Vogels, Tim P., Robert C. Froemke, Nicolas Doyon, Matthieu Gilson, Julie S. Haas, Robert Liu, Arianna Maffei et al. 2013. Inhibitory synaptic plasticity: spike timing-dependence and putative network function. *Frontiers in neural circuits* 7.

Wang, Xiaobin, Yiran Chen, Hai Li, Dimitar Dimitrov, and Harry Liu. 2008. Spin torque random access memory down to 22 nm technology. *IEEE Transactions on Magnetics* 44(11):2479-2482.

Warren, R., J. Reifenberg, and K. Goodson. 2008. Compact thermal model for phase change memory nanodevices. *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on* 1018-1045.

Wei, Xuechao, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs. *Proceedings of the 54th Annual Design Automation Conference 2017* 29.

Wong, H-S. Philip, Simone Raoux, SangBum Kim, Jiale Liang, John P. Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E. Goodson. 2010. Phase change memory. *Proceedings of the IEEE* 98(12):2201-2227.

Wong, H-S. Philip, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T. Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM." *Proceedings of the IEEE* 100(6):1951-1970.

Xia, Lixue, Peng Gu, Boxun Li, Tianqi Tang, Xiling Yin, Wenqin Huangfu, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. 2016. Technological exploration of rram crossbar array for matrix-vector multiplication. *Journal of Computer Science and Technology* 31(1):3-19.

Xu, Zihan, Ketul B. Sutaria, Chengen Yang, Chaitali Chakrabarti, and Yu Cao. 2012. Hierarchical modeling of phase change memory for reliable design. *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, 115-120.

Xu, Zihan, Ketul B. Sutaria, Chengen Yang, Chaitali Chakrabarti, and Yu Cao. 2013. Compact modeling of STT-MTJ for SPICE simulation. *Solid-State Device Research Conference (ESSDERC), 2013 Proceedings of the European*, 338-341.

Xu, Zihan, Chengen Yang, Manqing Mao, Ketul B. Sutaria, Chaitali Chakrabarti, and Yu Cao. 2014. Compact modeling of STT-MTJ devices. *Solid-State Electronics* 102:76-81.

Xu, Zihan, Abinash Mohanty, Pai-Yu Chen, Deepak Kadetotad, Binbin Lin, Jieping Ye, Sarma Vrudhula, Shimeng Yu, Jae-sun Seo, and Yu Cao. 2014. Parallel programming of resistive cross-point array for synaptic plasticity. *Procedia Computer Science* 41:126-133.

Xu, Zihan, Pai-Yu Chen, Jae-sun Seo, Shimeng Yu, and Yu Cao. 2016. Hardware-efficient learning with feedforward inhibition. *Nanoelectronics Conference (INEC), 2016 IEEE International*, 1-2.

Xu, Zihan, Steven Skorheim, Ming Tu, Visar Berisha, Shimeng Yu, Jae sun Seo, Maxim Bazhenov, and Yu Cao. 2017. Improving efficiency in sparse learning with the feedforward inhibitory motif. *Neurocomputing*.

Yang, Chengen, Yunus Emre, Yu Cao, and Chaitali Chakrabarti. 2012. Multi-tiered approach to improving the reliability of multi-level cell PRAM. *Signal Processing Systems (SiPS), 2012 IEEE Workshop on* 114-119.

Yu, Jun, Xiaokang Yang, Fei Gao, and Dacheng Tao. 2016. Deep multimodal distance metric learning using click constraints for image ranking. *IEEE transactions on cybernetics* 99:1-11.

Yu, Jun, Jitao Sang, and Xinbo Gao. 2017. Machine learning and signal processing for big multimedia analysis. *Neurocomputing* 257:1-4.

Yu, Jun, Baopeng Zhang, Zhengzhong Kuang, Dan Lin, and Jianping Fan. 2017. iPrivacy: image privacy protection by identifying sensitive objects via deep multi-task learning. *IEEE Transactions on Information Forensics and Security* 12(5):1005-1016.

Yu, Shimeng, Bin Gao, Zheng Fang, Hongyu Yu, Jinfeng Kang, and H- S. Philip Wong. 2013. A low energy oxide- based electronic synaptic device for neuromorphic visual systems with tolerance to device variation. *Advanced Materials* 25(12):1774-1779.

Zhang, Jian, Ke Li, Yun Liang, and Na Li. 2017. Learning 3D faces from 2D images via Stacked Contractive Autoencoder. *Neurocomputing* 257:67-78.

Zhao, Ritchie, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani B. Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs." *Field-Programmable Gate Arrays FPGA* 15-24.

Zhao, Wei, and Yu Cao. 2006. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on Electron Devices* 53(11):2816-2823.

Zhou, Shuchang, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv* preprint arXiv:1606.06160.

Zylberberg, Joel, Jason Timothy Murphy, and Michael Robert DeWeese. 2011. A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields. *PLoS computational biology* 7(10):e1002250.