Neuromorphic Computing with Resistive Switching Devices

by

Patrick M. Sheridan

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in the University of Michigan
2015

Doctoral Committee:

Associate Professor Wei Lu, Chair
Professor L. Jay Guo
Professor Jerome P. Lynch
Associate Professor Zhengya Zhang

Table of Contents

List of Figures

Abstract

Resistive switches, commonly referred to as resistive memory (RRAM) devices and modeled as memristors, are an emerging nanoscale technology that can revolutionize data storage and computing approaches. Enabled by the advancement of nanoscale semiconductor fabrication and detailed understanding of the physical and chemical processes occurring at the atomic scale, resistive switches offer high speed, low-power, and extremely dense nonvolatile data storage. Further, the analog capabilities of resistive switching devices enables neuromorphic computing approaches which can achieve massively parallel computation with a power and area budget that is orders of magnitude lower than today's conventional, digital approaches.

This dissertation presents the investigation of tungsten oxide based resistive switching devices for use in neuromorphic computing applications. Device structure, fabrication, and integration are described and physical models are developed to describe the behavior of the devices. These models are used to develop array-scale simulations in support of neuromorphic computing approaches. Several signal processing algorithms are adapted for acceleration using arrays of resistive switches. Both simulation and experimental results are reported. Finally, guiding principles and proposals for future work are discussed.

Chapter 1. Introduction

1.1 The von Neumann Bottleneck

In 1945, John von Neumann proposed [1] a computing architecture that proscribed separating program and data memory from arithmetic and logical computations. Instructions and operands are to be fetched from memory, a computation performed in the arithmetic-logic unit (ALU), and the results returned to memory. Broadly speaking, the stored program paradigm has been used in nearly all computing systems to date due to its ease of programming and intuitive operation. The von Neumann architecture, however, suffers from a fundamental drawback: the separation of memory and computing elements requires a constant movement of data across a finite-width bus (or several busses) in order to perform operations, and this movement requires significant energy and time expenditures [2].

An alternative computing approach is found in biological systems which must operate on a highly constrained power budget. For example, the human brain, arguably the most powerful computer for certain tasks, is estimated from blood flow measurements to perform all of its functions while using approximately 35 watts [3]. It is believed that the brain accomplishes this feat by approximating computational tasks in the analog domain and by integrating the memory and computational elements, thereby avoiding the von-Neumann bottleneck [4], [5]. The details of how this is accomplished remain an open question in the fields of neuroscience and computational biology; nonetheless, insights

into the functioning of the brain have inspired new computing paradigms, termed "neuromorphic computing" that can achieve significant performance and efficiency over traditional computer organizational designs [6]–[8].

1.2 Neuromorphic Computing

Carver Mead coined the term "neuromorphic" in the early 1990s to describe a neural information processing paradigm that is fundamentally different, and orders of magnitude more energy efficient, from digital computation. A fundamental principle of neuromorphic computing is to use the governing physics of a device to perform computation, rather than using the device merely as a digital switch as is done with transistor logic today [9], [10]. Furthermore, a defining characteristic of neuromorphic systems is that they use distributed memory and computational elements [11]. This paradigm eliminates the von Neumann bottleneck, described above, enabling massively parallel computation while drastically reducing the energy required to shuttle data to and from storage elements to an arithmetic-logic unit where it can be used. Resistive switches are emerging devices that provide a key technology that allows designers to combine memory and computing elements to save space and energy. This reorganization of the computing datapath requires a rethinking of computer architectures and the algorithms they run. Conventional algorithms with sequential processing steps, resulting in complex data dependencies, are often not well suited to massively parallel, distributed computing systems [12]. This dissertation will examine both novel hardware architectures constructed using resistive switching elements and neuromorphic algorithms that are designed to use the new hardware efficiently.

In the proposed approach, resistive switching elements are used to both store analog weights and directly perform information processing. The system relies on the intrinsic hysteresis of the devices to perform an accumulation function of weight updates during learning. The variable resistance of the device is used to modulate a current signal, effectively performing a multiplication operation. Finally, we rely on the crossbar structure of arrays of devices, combined with Kirchoff's current law, to sum current signals from multiple devices to calculate a weighted sum. Because of the limited precision inherent in analog systems, we have targeted applications that can tolerate imprecision and occasional inaccuracy. Additionally, we rely heavily on feedback mechanisms to self-correct errors. The following section will describe the characteristics of resistive switching devices in more detail before system-scale structures are examined for computing.

1.3 Resistive Switching

Resistive switching phenomena has been investigated as a possible successor to Flash memory technologies for non-volatile data storage due to its high packing density, stackability, speed, and low power operation. While hysteretic resistors have been studied at least since the 1960s [13], [14], interest has surged as advances in lithography and semiconductor fabrication techniques have reduced device dimensions to the nanoscale which has improved resistive switching device characteristics [15], and our group has been working on resistive switching devices since 2005 [16], [17]. In 2008, HP researchers connected the hysteretic behavior of titanium-dioxide-based resistive switches with the theoretical framework of a memristor [18], a device postulated to be the

fourth fundamental circuit element by Leon Chua in 1971 [19]. A memristor is a two terminal device wherein the conductance between the two terminals is a function of the history of signals applied to the device. The hysteretic behavior gives rise to the characteristic pinched current voltage loop as shown in Figure 1.

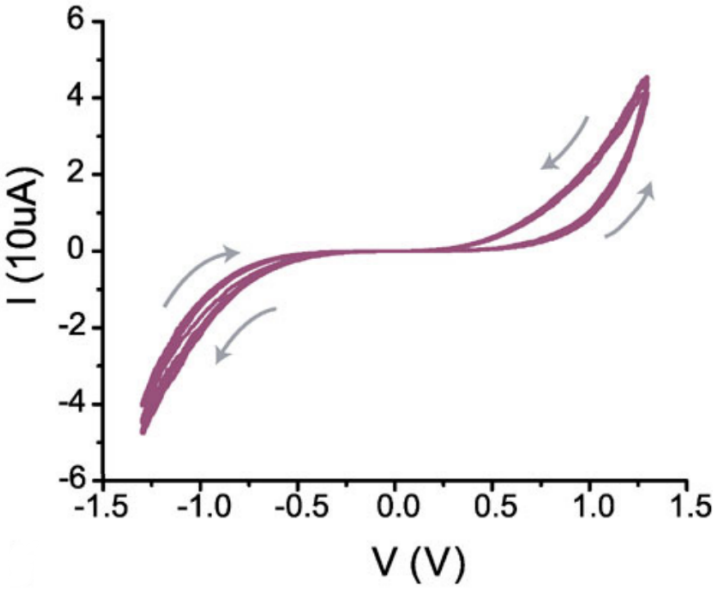


Figure 1.1: Hysteresis loop characteristic of resistive switching devices.

The physical manifestion of a memristor can be created by sandwiching an electrically switchable resistance layer between two electrodes. A three-dimensional representation of such a switch is shown in Figure 1.2 below.

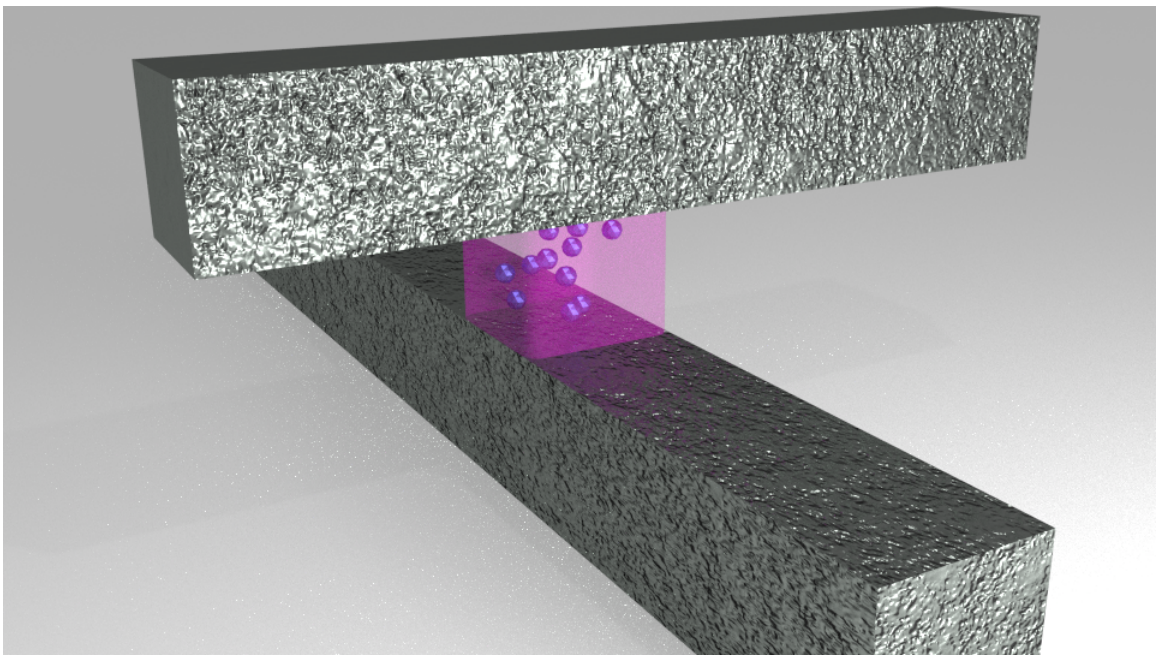


Figure 1.2: A three-dimensional representation of a generic resistive switch. A resistive switching layer (shown in pink) is sandwiched between two conductive electrodes (shown in grey). Conductance modulation is achieved by rearranging the conductive elements (shown in blue).

For a voltage-controlled resistive switch, a voltage is applied between the top and bottom electrodes. The resultant current provides an electrical signal that is used to probe the state of the device. If the voltage is of sufficient magnitude, it will simultaneously cause a measurable change in the state of the device. This is accomplished by causing the rearrangement of the conductive elements (depicted as blue spheres in Figure 1.2) within the otherwise insulating layer (depicted in pink). The arrangement of the conductive element does not change instantaneously and normally can maintain for a very long time even after the applied voltage is removed. This allows the new state to be 'memorized' and is the origin of the observed hysteretic behaviors. In this sense, the device 'remembers' the history of applied voltages. In the absence of an applied voltage signal, the current output will be zero (hence the resistive switch will always have a zero

crossing—or pinched—hysteresis curve as shown in Figure 1.1). This is an important feature of a resistive switch or memristor: unlike a capacitor or inductor, the device does not store energy as an electric charge or magnetic field (except via parasitics). The nature of the resistive switching layer and the conductive elements is discussed in more detail in Chapter 2.

1.3.1 Competing Technologies

Resistive switches (RRAMs) are not alone in the competition for neuromorphic-enabling technologies. Approaches such as DRAM and SRAM offer an alternative, with the latter even achieving commercial success in high-speed content addressable memories [20], but suffer from relatively large area and power requirements [21]. Furthermore the volatile nature of DRAM and SRAM limits power efficiency, particular in applications that require only intermittent operations. Flash memory, by contrast, offers high density, non-volatile storage. The recent emergence of commercial 3D NAND Flash [22], [23] has extended the usefulness of the technology and its continued development offers exciting possibilities for data storage. However, high programming voltages, long write times, block-erase, and limited endurance limit the technology [24] in computing applications. Phase change memory (PCM) is another viable technology that offers many of the same advantages as RRAM, namely high density, non-voltage storage and random access. For this reason PCM has seen some commercial success [25] but is limited by a power intensive write process [26] which limits its density due to thermal coupling among neighboring cells, and the use of exotic materials which affects its manufacturability and cost/bit. Additionally, while a PCM cell can be used to store multiple bits, its resistance cannot be easily incrementally adjusted, particularly during

the erase process. For these reasons, RRAMs offer the greatest promise for enabling scalable, reliable, low-power neuromorphic computing.

1.4 Organization of Dissertation

The first chapter discusses resistive switching device characteristics and physical mechanisms of operation. A framework for device modeling is developed both at the single cell SPICE level and for larger arrays of memristive devices. A framework for implementing network scale learning architectures is outlined and used in the chapters that follow. Chapter 2 discusses the test and measurement setups that have been developed to support data collection and network implementation. Integration with CMOS circuits is demonstrated as well as larger arrays used for network learning.

Chapters 3 and 4 review network-level learning approaches that make use of resistive switching devices for accelerated computation. Chapter 3 examines a common machine learning task, vector quantization, while Chapter 4 investigates the use of memristors to develop a sparse coding accelerator for natural images.

The next two chapters discuss the implementation of computation acceleration with real RRAM devices formed in a crossbar array. Chapter 5 discusses the test and measurement systems that were constructed as part of the dissertation work including an array measurement platform and CMOS chip integration. Chapter 6 makes use of these systems to experimentally show a sparse coding algorithm's implementation and acceleration using resistive switching hardware.

Finally, the dissertation draws conclusions in Chapter 7. A section is included to discuss possible future directions for this and related research. This includes the implementation of in-situ network learning for sparse dictionary construction as well as

new approaches to using resistive switching crossbar hardware. The final chapter includes all references found in the dissertation.

Chapter 2. Device Simulation

Device modeling plays an important role in motivating and directing experimental research with RRAM devices (memristors). In order to facilitate accurate device and array simulations, several models were developed with differing levels of speed and accuracy. Network level simulations were implemented both in the industry standard Simulation Program with Integrated Circuit Emphasis (SPICE) and a custom simulation framework. The custom framework allowed the comparatively rapid exploration of the design space and testing of learning parameters while sacrificing limited fidelity to actual network operation. The first model focused on cation conduction in a solid electrolyte, also known as electrometallization cell or conductive bridging RAM. An additional model was developed for anionic devices based on oxygen vacancy movement. To explore learning algorithms in larger memristive networks, a simulation framework was developed in Python.

2.1 SPICE Modeling

Two-terminal resistive switches, often termed memristors [18], [19], [27], [28], are electronic devices that exhibit hysteretic resistance switching behavior in their *I–V* characteristics and have been proposed in a broad range of applications including, but not limited to, resistive random access memory (RRAM) [29]–[31], neuromorphic systems [32], Boolean logic implementation [33], signal processing, and circuit design [34]. Such circuits can extend the functional scaling of integrated circuits beyond CMOS, and offer

non-volatility and 3D integration potential [35], [36]. In particular, as a potential replacement for Flash memory technology, RRAM has generated significant interest in ultra-high density non-volatile information storage applications. A broad range of materials have been studied that can act as RRAM devices [37]–[40].

On the other hand, there has been a lack of well-established models that can simulate and predict the resistance switching effects observed in RRAM devices. Previous attempts to simulate RRAM memory or circuits have either focused entirely on the steady state, with fixed resistances assigned to the devices, or used a fixed threshold voltage, fixed switching time and predetermined on-resistance R_{on} (*e.g.* by using a voltage controlled switch to emulate an RRAM device). Unfortunately, these approaches do not correctly capture the critical dynamic switching properties of RRAM devices. In particular, previous experimental studies have shown that the threshold voltage, switching time, and R_{on} are not fixed parameters but rather are dynamic effects and vary with differing circuit conditions even for the same device [41], [42]. In this section, we discuss the development of a physics-based device model that can accurately predict the dynamic effects during resistance switching. In addition, we show the analytical model can be incorporated into standard circuit simulators such as SPICE, by creating a subcircuit and using floating node voltages to represent the internal state variables. Such a SPICE model can accurately predict the switching characteristics of the RRAM device, such as the dependence of switching time on voltage, the apparent threshold effect and its dependence on sweep rate, and the multi-level storage effect. The development of the device and SPICE model will greatly aid the simulation and design of memory and logic

circuits based on RRAM devices. Furthermore, the framework developed here can be applied to a broad range of resistance switching devices.

2.1.1 Cationic Devices

A memristor model for cationic devices based on the field-driven movement of metal ions in an insulating matrix was developed in SPICE. The research resulted in the publication of a journal article in *Nanoscale* [43]. At the time of publication, most memristor models [18], [44] were phenomenological in nature, with a number of free parameters to allow fitting of observed data. In developing our model, we sought to use physical constants, obtained either through experiment or available literature, to simulate device behavior from a first principles approach.

The model is based on a typical cation RRAM device structure consisting of a bottom palladium electrode and a top silver (Ag) electrode sandwiching a switching layer of amorphous silicon (α-Si). By applying a positive voltage to the silver electrode, while holding the bottom electrode at ground, silver atoms in the electrode near the α-Si interface can be oxidized and dissolve in the α -Si. The Ag cations can then hop between defects in the α-Si matrix and the accumulation of the Ag cations eventually forms a Ag conductive filament allowing the conduction of electrons through the otherwise insulating α-Si. This is shown schematically in Figure 2.1. A voltage of the opposite polarity can be used to reverse the process and return the device to a highly resistive state.

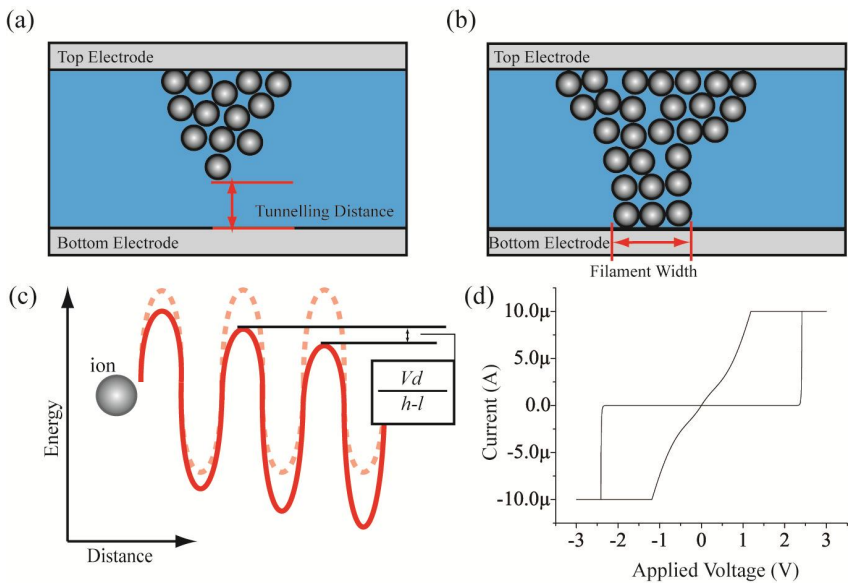


Figure 2.1: Filament growth dynamics. Schematics of filament length (a) and width (b) growth. (c) Energy potential seen by a metal ion in the insulating matrix. (d) Resulting hysteresis curve (with current compliance).

The disassociation and hopping of Ag ions is primarily a field-driven process. After fabrication, a voltage on the order of ~5V is necessary to form the devices. During this electroforming process, relatively large amounts of Ag cations are injected into the α-Si, decreasing the gap between the top and bottom electrodes. Subsequent voltage pulses of lower amplitude can then be used to switch the device since the filament only needs to bridge a smaller gap after the initial forming process. Joule heating is also thought to play a role in switching by increasing the local temperature, exponentially expediting the Ag ion hopping process.

The dynamics of the ‘‘filaments’’ will in turn be determined by the motion of their constituent ions. As a first example, we consider a growth model where the state variable, renamed l, represents the length of a conductive filament. With the application

of a positive bias to the top electrode while keeping the bottom electrode grounded, filament growth can be initiated inside the insulating material. The filament body is assumed to be metallic and with low resistance and thus, to model the growth of the filament, we need only consider the motion of the leading ion. For simplicity, it is further assumed that the ion moves in one dimension—parallel to the applied electric field. The growth rate of the filament is then determined by the ‘‘drift’’ speed of the leading ion, which can in turn be derived by calculating how long it takes for the ion to hop over an energy barrier to reach to a new site, schematically illustrated in Fig. 2.1 (c) and given by

$$\frac{dl}{dt} = dv\left(\exp\left(\frac{-qU_a + qVd/2(h-l)}{kT}\right) - \exp\left(\frac{-qU_a - qVd/2(h-l)}{kT}\right)\right) \quad (2.1a)$$

which can be expressed equivalently as

$$\frac{dl}{dt} = 2dv\exp\left(\frac{-qU_a}{kT}\right)\sinh\left(\frac{qVd}{2kT(h-l)}\right) \quad (2.1b)$$

where

l is the filament length

d is the inter-site hopping distance

v is a characteristic attempt frequency

k, T, q is Boltzmann’s constant, temperature, and electron charge respectively

U_a is the activation energy

h is the total gap height between the post-forming top electrode and bottom electrode

We can intuitively understand Eqn. (2.1) as follows: the particle’s ‘‘drift’’ velocity is a product of the distance travelled in each hop, d, and the frequency with which these hops occur. The latter is given by the attempt frequency, v, scaled by a factor

exponentially dependent on the apparent barrier height since the hopping process is thermally activated. Under an applied bias, the apparent barrier height will be reduced from the barrier at zero-bias, U_a , to U_a-$Ed/2$, where E is the local electric field, as schematically shown in Fig. 2.1(c). Assuming the voltage is dropped linearly along the distance between the filament tip and the opposing electrode $(h-l)$, the apparent barrier seen by the ions will then be lowered by $qVd/2(h-l)$. As a result, the filament growth rate will be enhanced exponentially as a function of the applied voltage. This is contrary to the typical linear drift-field relationship, which can be considered as a low-field approximation. The exponential growth of the filament as a function of the applied field (Eq. 2.1) is a key characteristic of RRAM devices and enables the device to be programmed quickly (within 10ns) at high field while maintaining very long data retention after the field (voltage) is removed.

The second exponential term in Eqn. (2.1a) is included to account for the probability that the particle will hop backwards, towards the originating electrode. On the other hand, one should note that Eqn. (2.1) is limited to the voltage ranges such that U_a $Vd/2(h-l) > 0$. At very high field, such as when V is very large or $(h-l)$ is very small, U_a $Vd/2(h-l) < 0$, a result that is not physical. As a consequence, Eqn. (2.1) will overestimate the filament growth rate at high field. In addition, under these high-field conditions, the filament growth will instead be dominated by other processes such as the oxidation rate of the metal atoms, rather than the field at the tip of the filament. At high biases, it is thus reasonable to re-write Eqn. (2.1) as

$$\frac{dl}{dt} = s \exp\left(\frac{-qU_a}{kT}\right) \sinh\frac{V}{V_0} \tag{2.2}$$

where V_0 and s are treated as free parameters to replace the previous free parameters v and d.

The exponential dependence of the filament growth on V (Eq. 2.2) suggests that the filament growth is a self-limiting process. As the filament grows and the device becomes more conductive, progressively more voltage is dropped on the resistance in series with the device. With less voltage across the device, the filament growth is dramatically slowed as dictated by the exponential dependence of the growth rate on V. In this manner, the final state of the device can be controlled by changing the resistance in series with the device, which is normally implemented through a current compliance. Simulation results for this self-limited growth effect are shown in Figure 2.2 below.

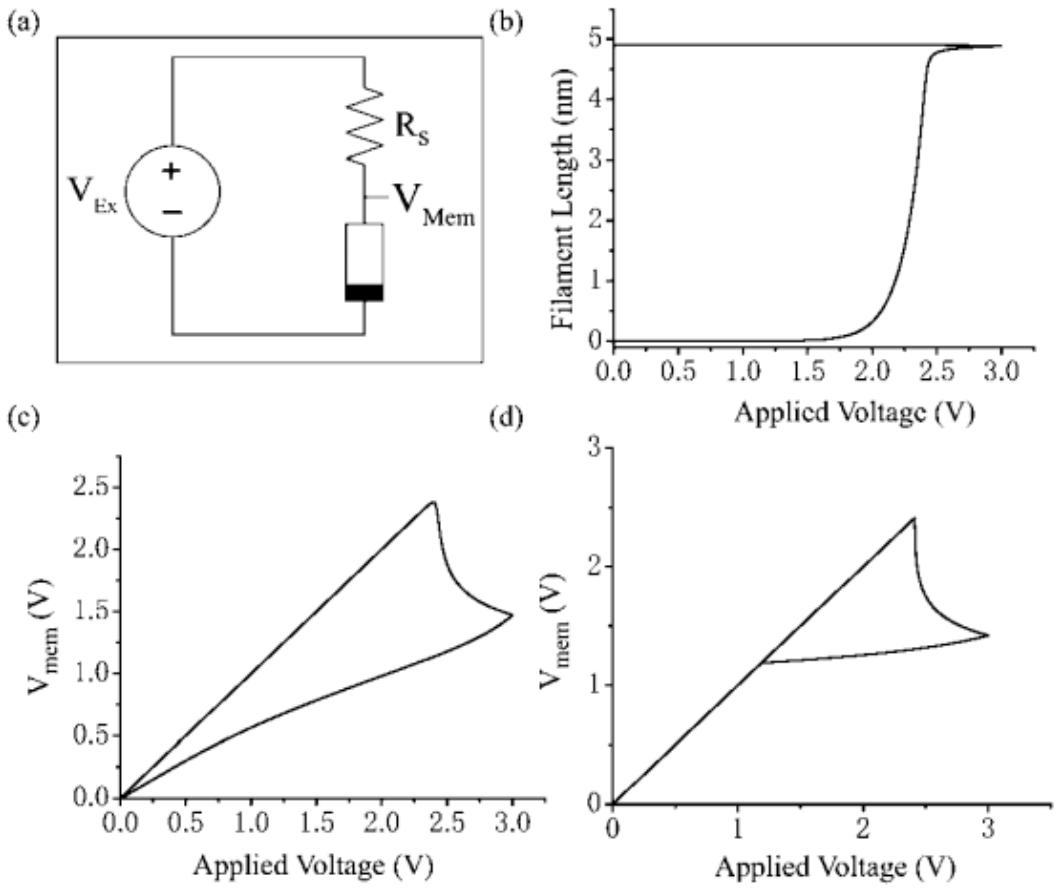


Figure 2.2: DC voltage sweep simulations. (a) Circuit schematic. (b) Filament growth during the DC sweep. (c) The voltage seen by the resistive switch. As the filament grows and the device becomes more conductive, progressively more voltage is dropped on the series resistor. (d) Similar feedback effect is obtained by using the current compliance instead of a series resistor.

The state-variable, filament length *l* (or filament width *w*), was stored as a voltage across a floating capacitor as done in [44]. A schematic of how this is implemented is shown in Figure 2.3. The model was later translated into Verilog-AMS, which allowed for a more direct coding of the model, while maintaining the ability to compile it for use with SPICE.

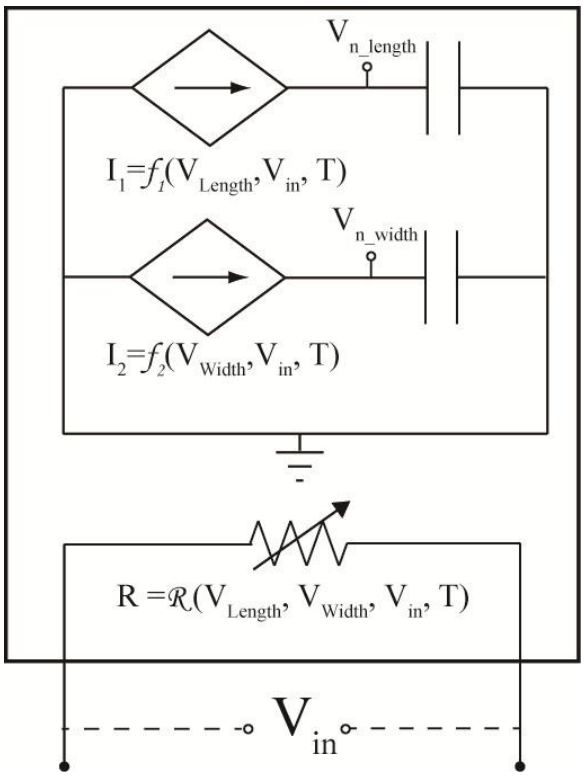


Figure 2.3: SPICE implementation of RRAM component. State variable parameters are stored on capacitors. Reproduced from [43].

The next step is to determine the I–V relationship of the device for Eqn. (2.1) or (2.3). Following the arguments above, in an RRAM device where the filament has not bridged the electrodes, the resistance will be dominated by that between the tip of the filament and the opposing electrode with a distance expected to be on the order of a few nanometres. At such distances, it is reasonable to assume that current is dominated by tunneling [45]. Using the expressions for a square barrier obtained from [46] the current can be expressed as:

$$I = A\frac{4q\pi m(kT)^2}{h_0^3}\exp(-b_1)\frac{1}{(c_1kT)^2}\frac{\pi c_1 kT}{\sin(\pi c_1 kT)}(1-\exp(c_1qV) \qquad (2.3)$$

where

$$b_1 = \begin{cases} \frac{2\alpha(h-l)\sqrt{q}}{3V}\left(\phi_0^{3/2} - (\phi_0 - V)^{3/2}\right) & \text{if } V < \phi_0 \\ \frac{2\alpha(h-l)\sqrt{q}}{3V}\phi_0^{3/2} & \text{if } V > \phi_0 \end{cases},$$

$$c_1 = \begin{cases} \frac{\alpha(h-l)}{V\sqrt{q}}\left(\phi_0^{1/2} - (\phi_0 - V)^{1/2}\right) & \text{if } V < \phi_0 \\ \frac{\alpha(h-l)}{V\sqrt{q}}\phi_0^{1/2} & \text{if } V > \phi_0 \end{cases},$$

A is the filament area, m is the effective electron mass, h_0 is Planck's constant, and ϕ_0 is the barrier height at zero applied bias and

$$\alpha = \frac{2\sqrt{2m}}{\hbar}$$

To reduce the computational complexity for modeling purposes, the tunneling expression was simplified to

$$I = Aq\frac{8\pi^2 m}{h_0^3}\frac{kT}{c_1 \sin(\pi c_1 kT)} \quad V < \phi_0 \tag{2.4a}$$

$$I = A\frac{4\pi q^2 m}{h_0^3 \alpha^2 \phi_0}\left(\frac{V}{h-l}\right)^2 e^{-\frac{2\alpha\sqrt{q(h-l)\phi_0^{3/2}}}{3V}} \quad V > \phi_0 \tag{2.4b}$$

Figure 2.4 shows comparisons of results obtained from the simplified (Eqn. (2.3)) and the full expression (Eqn. (2.4)), illustrating the accuracy of the simplification

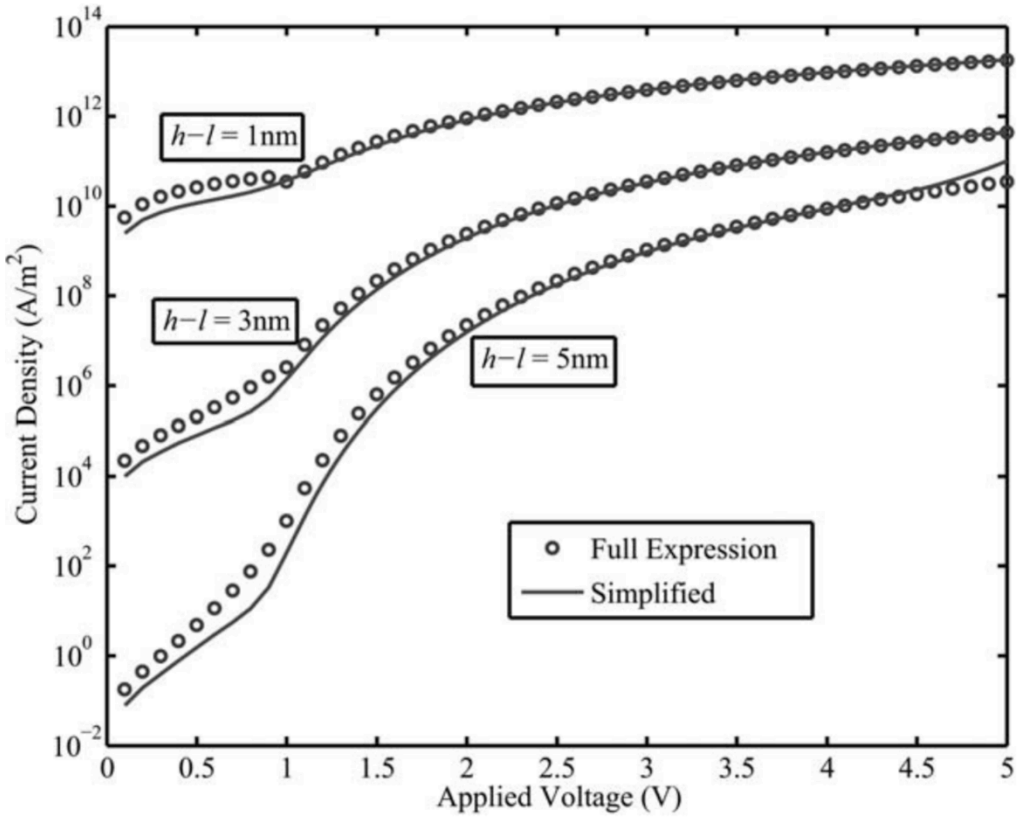


Figure 2.4: Comparison between the full tunneling expression (eqn (2.3)) and the simplified, smoothed function (eqn (2.4)) at different tunneling gap (*h-l*) conditions. The two functions agree well in the voltage range of interest, allowing us to use the simpler expression without losing accuracy.

By self-consistently solving the filament grow dynamic equation (Eqn. 2.2) and the I-V equation (Eqn. 2.4) the dynamic resistive switching model can be established. Simulations can then be used to predict device behaviors as shown in Figs. 2.5 and 2.6 below:

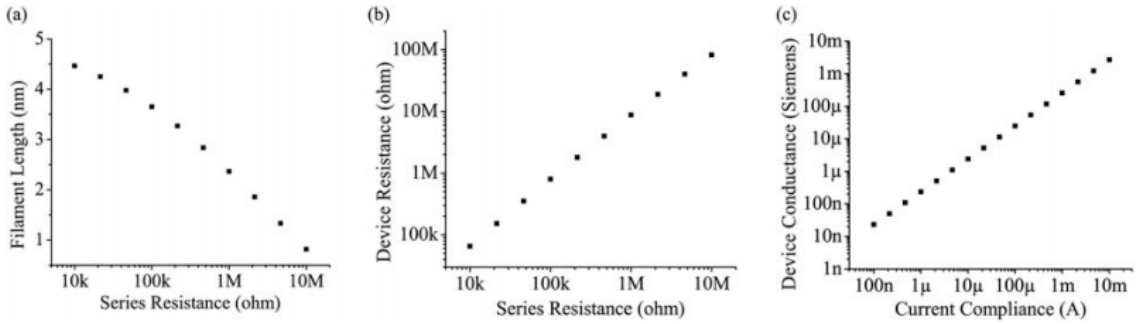


Figure 2.5: Multi-level programming obtained in RRAM cells. (a) Dependence of the filament length l on the series resistance RS. (b). Dependence of the final device resistance R_{on} on the series resistance R_S, plotted in log–log scale. (c) Dependence of the R_{on} on the compliance current, plotted in log–log scale

The final resistance of the device can be controlled using either a series resistance or the compliance current, as shown in Fig. 2.5. The ability to reliable control device resistance allows for the multiple bits of information to be stored in a single cell (termed multi-level cell—MLC RRAM).

The device model can also be used to predict the switching time (defined to be when the current exceeds 1μA) as shown in Fig. 2.6. There is an exponential relationship between applied voltage and switching time (Fig. 2.6(b)). The model allows circuit designers to determine necessary programming voltages to achieve a desired programming speed. Furthermore, this model has been extended [47] to include stochastic switching effects and the relationship between voltage and switching time provides a means to build a biased random number generator for stochastic computing. Finally, the equations also explain the apparent threshold voltage in device switching and demonstrate that there is not a fixed threshold, but that it is dependent on voltage sweep rate as shown in Fig. 2.7

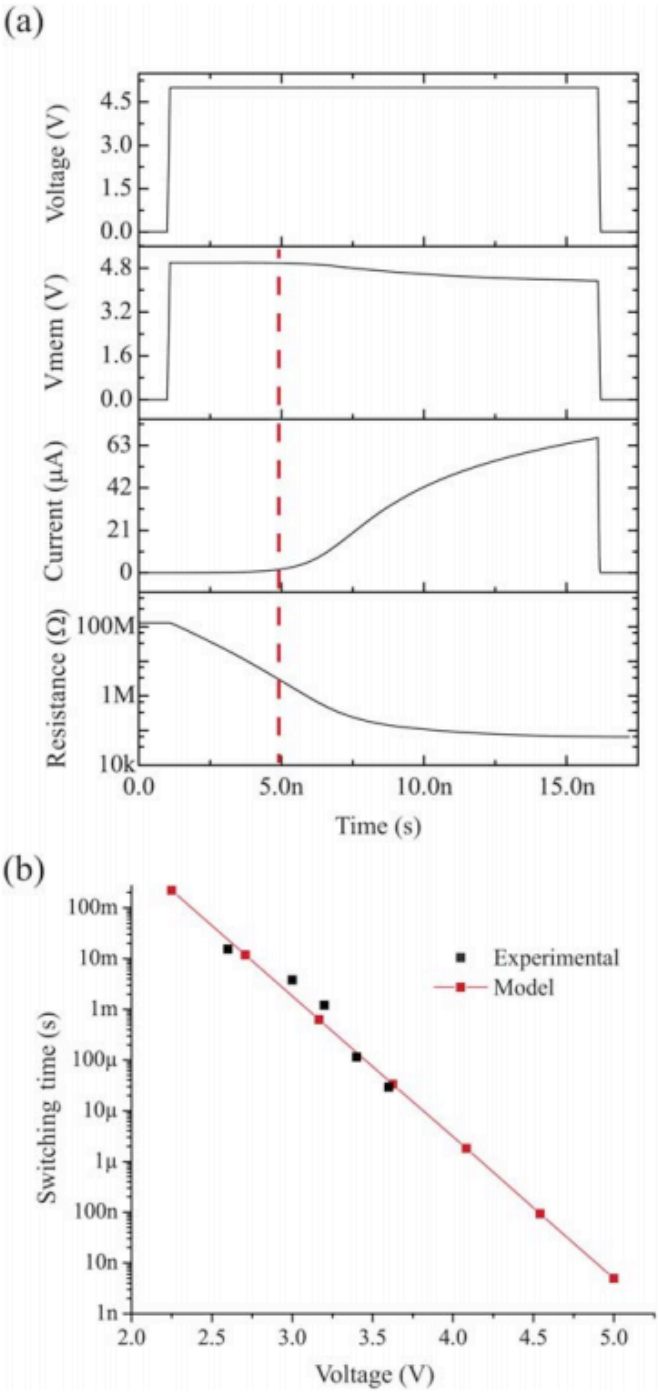


Figure 2.6: (a) RRAM dynamics during pulse programming. The circuit in Fig. 2.2(a) is used with $R_s = 10k\Omega$. Top to bottom: applied voltage, voltage across the RRAM cell, current, and device resistance. The switching event is defined as when the current is >1μA and is marked by the dotted line. (b) Dependence of the switching time on applied voltage.

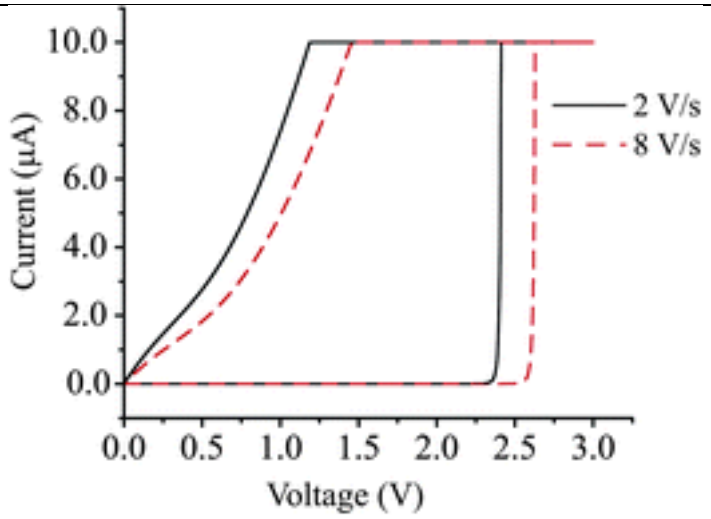


Figure 2.7: Switching characteristics with two different sweep rates. The apparent threshold voltage is dynamic and dependent upon the sweep rate, with a faster sweeping rate resulting in a larger "threshold voltage."

2.1.2 Anionic Devices

Resistive switching devices based on anion (typically in the form of oxygen vacancy, V_O) motion form another class of RRAM devices. Here the V_Os in transition metal oxides, such as TiO_x, TaO_x, HfO_x, WO_x, NiO_x, and AlO_x [40], [48] act as donors in the material and the accumulation (depletion) of V_Os leads to the increase (decrease) of the device conductance. Additionally, by incrementally changing the local V_O concentration, the conductance can be modulated incrementally allowing the device to exhibit analog switching behavior. As shown in Figure 2.8b-d, gradual resistance changes can be obtained, in contrast to the abrupt (digital) resistance switching behaviors exemplified in Figure 2.1d.

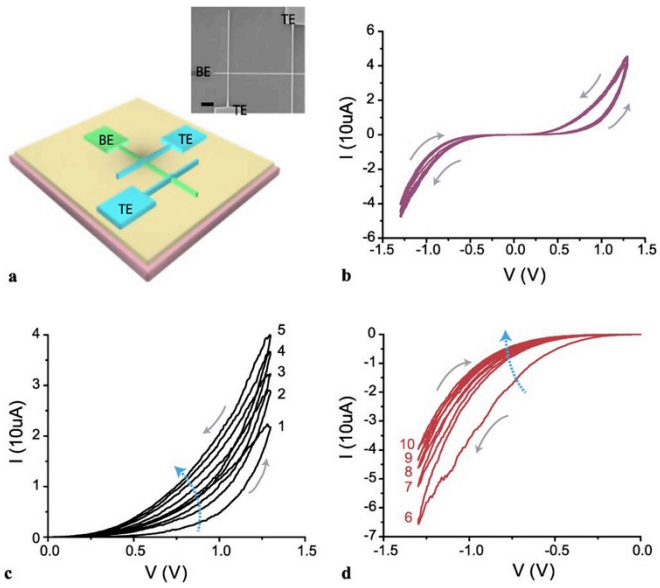


Figure 2.8: WO_x based anionic resistive switch. (a) Device structure (SEM inset). (b) Hysteresis curve. Repeated positive (c) and negative (d) voltage sweeps.

In these devices, the conducting path (filament) is considered to be a region rich in V_O. In analog resistive switching devices the V_O migration activation energy is low and multiple conductive paths can form in parallel [49]. The overall device conductance is then determined by the total area of the conducting regions, and the conducting area (scaled with the total device area) *w* was chosen as the state variable. The electron conduction will be dominated by tunneling (in the conducting region) or Schottky emission (in the non-conducting region) [49] and the total current includes both contributions weighted by the state variable *w*. The electronic current can then be modeled by the following equation:

$$I = w\,\gamma \sinh(\delta V) + (1 - w)\alpha(1 - \exp(-\beta V)) \tag{2.5}$$

The first term in the equation models the tunneling current in the conducting region, while the second term models the Schottky component in the non-conducting region. The state variable *w* represents the relative area of conductive over the total device area.

Like the cation-based devices, the operation of the anion-based devices critically depends on the dynamics of the state variable. The oxygen vacancy movement is driven by both drift in an applied electric field and spontaneous diffusion effects. This is modeled by the following equation:

$$\frac{dw}{dt} = \lambda \sinh(\eta_2 V) F(w, V) - \frac{w}{\tau} \tag{2.6}$$

The first term describes the non-linear drift of oxygen vacancies when a voltage *V*, is applied between the electrodes, very similar to Eq. (2.2) used for the cation-based devices. $F(w, V)$ serves as a window function to bound the growth of *w* in the physically meaningful range of [0,1]. The second term describes the tendency of oxygen vacancies to diffuse away from the conductive filaments, thus reducing the conductance of the device. The τ term is highly dependent on fabrication parameters including oxide thickness and quality. It has also been suggested that τ has a dependence on *w* which may result from an increased stability when vacancy concentration reaches a high enough concentration [50].

Similar to the cation-based devices, by self-consistently solving the filament dynamics equations (2.5) and the I-V equation (2.6), the hysteretic characteristics of the anion-based devices can be well-modeled, as shown in Fig. 2.9.

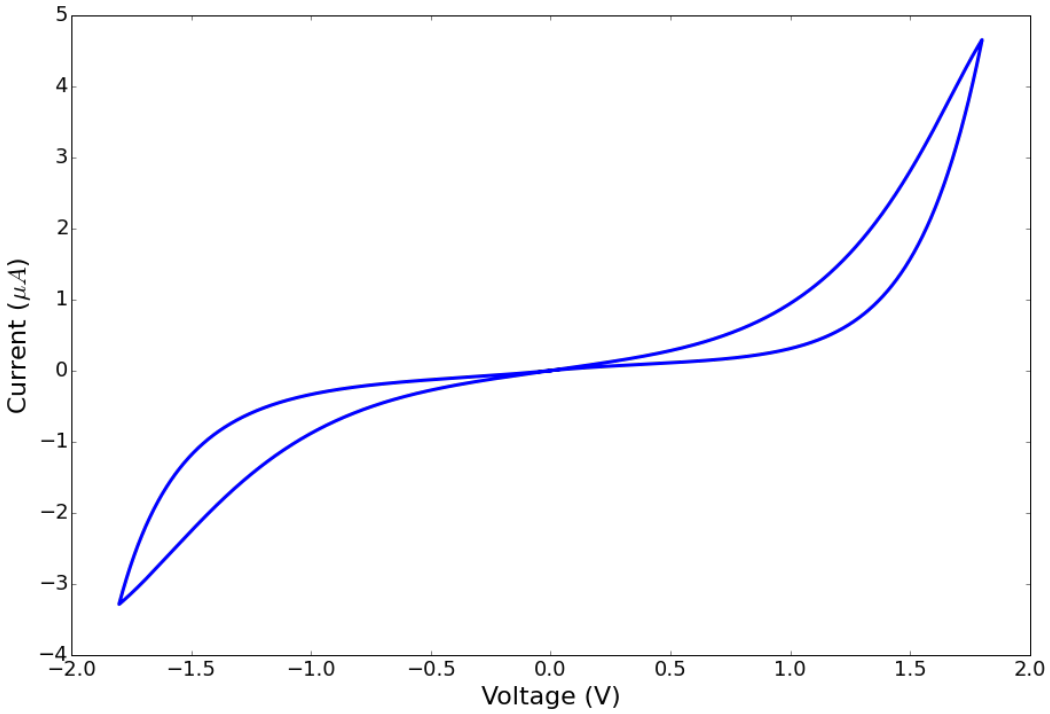


Figure 2.9: Hysteresis curve of a WO_x device. The gradual or analog conductance modulation is typical of anionic devices.

2.2 Array Simulation Framework

While SPICE simulation provides a very accurate picture of what is happening in the device, it requires significant computational effort, particularly as the network size grows. To make large network simulations more tractable, a simulation framework was built in Python. The network simulation framework was subsequently used in the simulation of RRAM-based neuromorphic systems discussed in Chapters 4 and 5. While the new framework uses the same device model developed for SPICE, some simplifying assumptions were made regarding array behavior which significantly reduced computational requirements. Specifically, it was assumed that array electrode resistances and sneak-path currents could be neglected. This allowed the current through each device to be solved analytically rather than simultaneously solving coupled equations

numerically, as is done by SPICE. The validity of these assumptions were verified with some long-running SPICE simulations and it was found that the direct approach proves adequate for algorithm development.

The simulation framework consists of a number of Python modules and classes that allow for multithreaded simulation with selectable degrees of realism. Network training and sparse coding can be run with several learning rules and coding algorithms using a pure software reference implementation, an ideal network using the device model, or a network with devices that incorporate parameter variations and/or noise.

An overview of the more significant units of the simulation framework is given below:

- An instance of the Python *simulation_object* class encapsulates all aspects relevant to a network simulation and stores the results of network training and algorithm executions within its fields.
- The *LCA* module performs the sparse coding of an input using the parameters from a simulation object.
- The *images_sim_utils* module encapsulates functions needed to generate training and testing samples as well as displaying the results from a sparse coding execution.
- Several training functions are defined within the *simulation* module that can be used for training the weights of a simulation_object. These functions cycle through the training samples provided and keep track of statistics about training operations. They are implemented so that they spawn a separate process so as to allow multiple simultaneous simulations.

- The training functions rely on the update functions, *Oja_update* or *gradient_descent*, to perform the calculations necessary for weight updates.

2.3 Conclusions

Physics-based models have been developed to describe the operations of cation-based and anion-based RRAM devices. Key to the model developments is the identification of the right state variable(s) and deriving the dynamic equations related to the state variable(s) and the associated I-V equations for given state variable(s). The device models and simulation framework have proved invaluable to the development of neuromorphic systems using these devices discussed in the following chapters. In larger arrays simulations certain aspects have been neglected (e.g. sneak-path current and device decay) after it was found from SPICE that these did not significantly impact network outcomes.

Continued device measurement has led to refined models that better incorporate temperature and vacancy diffusion effects. This has led to the development of so-called higher-order memristors with multiple state variables in a similar theoretical framework. The use of multiple state variables allows better emulation of the dynamic behaviors of biological synapses [51] and can improve the device switching characteristics. These recent results highlight that memristors are not just simply analog memory devices, but can actually offer rich internal dynamics that may lead to new neural circuit emulation or computational hardware developments beyond current approaches which emphasizes the network topology and weight.

Chapter 3. Learning Vector Quantization with Crossbars

Learning Vector quantization (LVQ) is a technique that maps an input vector to the nearest vector in a learned, stored dictionary [52]. The approach is a relatively simple form of lossy data compression with applications in signal processing and pattern recognition [53], [54]. Because of its reliance on a distance measure that is easy to compute with crossbar arrays, LVQ is a good candidate for implementation in resistive switching hardware. The following chapter will describe how LVQ can be implemented in a crossbar array. Simulation results demonstrating the feasibility of the approach will be shown and an experimental approach is outlined. Classification results from the MNIST handwritten data set are shown.

3.1 Description

Given a dictionary of stored vectors, Φ, and an input vector $\vec{p}$, vector quantization approximates $\vec{p}$ with $\vec{\phi}_w$ where $\vec{\phi}_w$ is the $\min_{\vec{\phi}} \|\vec{p} - \vec{\phi}\|$ and $\vec{\phi}$ are the columns of Φ. In the case of data compression, only the index of $\vec{\phi}_w$ in Φ needs to be recorded. In the classification task, the class of the input, $\vec{p}$, is assumed to have the same class as the winning dictionary element, $\vec{\phi}_w$. The approach divides the input space into regions of proximity to dictionary elements, with the boundaries occurring along a hyperplane that is equidistant from two dictionary elements. An example of this is shown graphically for the case of two dimensions in Fig. 3.1 below.

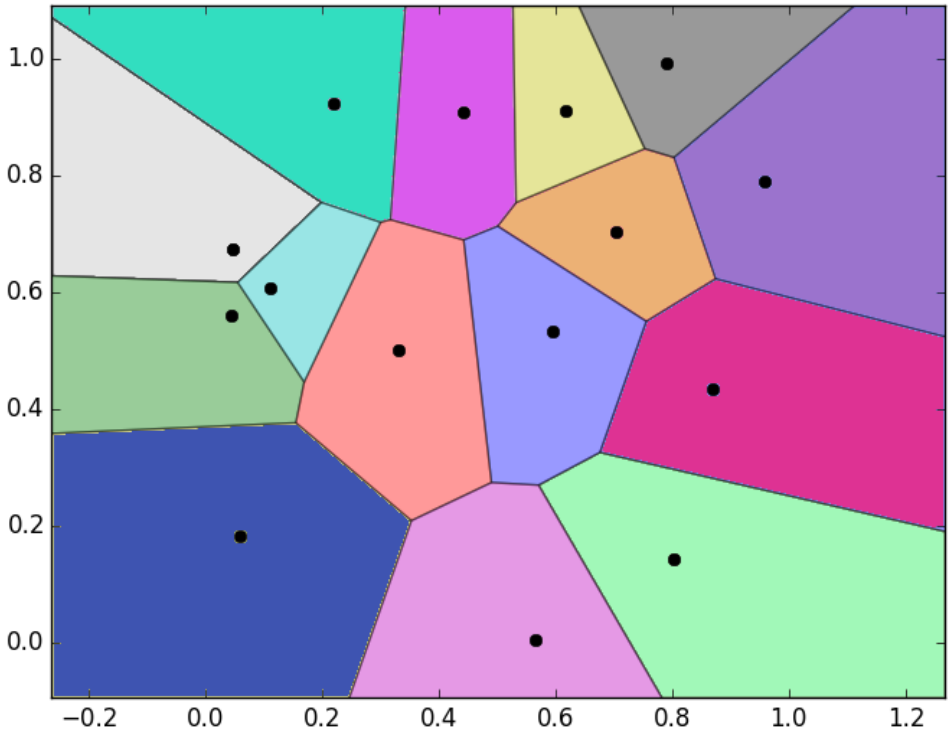


Figure 3.1: Voronoi diagram. The black dots represent dictionary elements while the black lines indicate equidistant boundaries between the dots. Any sample to be classified is assumed to have the same class as the black dot within the same region.

When an input is to be approximated (or classified), the nearest dictionary element (represented in black dots) is found and substituted for the original input (or its class used to determine the class of the input). The quality of the approximation depends on how well the dictionary elements divide the input space given the statistics of the inputs to be classified. The purpose of learning in LVQ is to distribute the dictionary elements to achieve a useful classification criterion.

3.2 Distance Measure

A key element of vector quantization is the ability to measure the distance of an arbitrary input to each element in the stored dictionary. If all of the elements in the

dictionary have the same L_2 (Euclidean) norm, then the distance comparisons can be easily accomplished using the dot-product operation as a result of the following relation:

$$\vec{p} \cdot \vec{\phi} = \|\vec{p}\| \left\|\vec{\phi}\right\| \cos\theta \tag{3.1}$$

where θ is the angle between $\vec{p}$ and $\vec{\phi}$.

Since $\vec{p}$ is shared between all comparisons, and if all $\vec{\phi}$ have the same norm, then

$$\min_{\vec{\phi}} \left\|\vec{p} - \vec{\phi}\right\| = \min_{\vec{\phi}} \theta = \min_{\vec{\phi}} \vec{p} \cdot \vec{\phi} \tag{3.2}$$

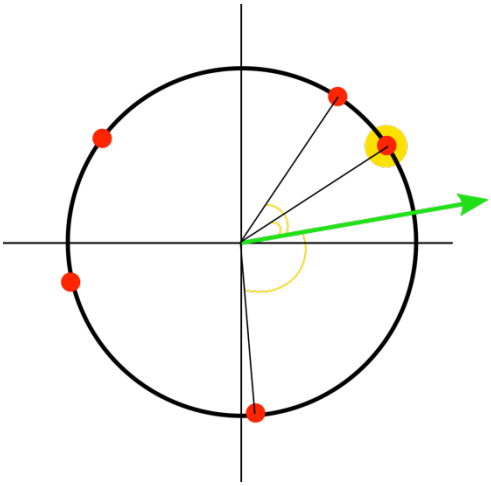


Figure 3.2: Graphical representation of vector quantization. An input vector (green arrow) is approximated as the nearest vector (yellow highlight) in a dictionary of stored vectors (red dots).

This gives us a straightforward way to find the nearest vector: we simply take the matrix-vector product, $\Phi\, \vec{p}$, which is simply the dot-product of $\vec{p}$ with each of the columns of Φ, and chose the index with the largest value. This is shown graphically for 2 dimensional inputs in Figure 3.2.

3.3 Crossbar Acceleration

An array of resistive switches can be used to accelerate the matrix-vector calculations. The following section will detail how this operation can be performed in an analog manner. The analog implementation allows computation to occur in memory, thus avoiding the von Neumann bottleneck. Additionally, a learning algorithm is presented that allows in-place update of memristor weights with the accumulation of update pulses performed by the intrinsic nature of the memristor.

First the values of the dictionary are stored as the analog conductances in a crossbar array of resistive switches. Each dictionary element, $\vec{\phi}$, which corresponds to a column of Φ, is stored element-wise in a column of the crossbar array. Specifically, the first element of $\vec{\phi}$ is stored in the first row of the column, and so on up to the last element which appears in the last row of the crossbar matrix. In an analogy to biological neurons in the visual cortex, $\vec{\phi}$ is considered to be the *receptive field* associated with the neuron connected to the column. Like biological receptive fields, the neuron's response to an input will be determined by how similar the input is to the neuron's receptive field; how this is accomplished is explained below.

The vector multiplicand, $\vec{p}$, is used to represent the inputs to the array, which are applied on the rows. Each element p_i of $\vec{p}$ is translated into an input pulse where the duration is linearly proportional to the magnitude of p_i, while the sign of p_i determines the input pulse polarity. The magnitude of the voltage for these input pulses is chosen to be of sufficient magnitude to allow the device state to be read, but not so large as to significantly modify the stored conductance; this is possible because of the strong non-

linearity of device state-change with respect to programming voltage discussed in Ch. 2. When the input pulse is no longer active, the corresponding row is connected to ground.

All of the column nodes are connected to virtual ground circuits (neurons) where the currents are collected and integrated. The charge that passes through an individual resistive switch is proportional to the product of the duration of the input pulse on its row electrode and its stored conductance value. Since all of the currents in a given column are summed (via Kirchoff's current law), the total charge integrated by the output neuron is proportional to the dot-product of the input and stored weights. Thus, the network has effectively performed vector-matrix multiplication in an analog-computing manner through a simple read process. Additionally, because the input pulses are shared among all of the columns of the crossbar array, all of the calculations occur in parallel, yielding another significant speedup.

3.4 Learning

Thus far, we have discussed how to perform vector quantization assuming we have a dictionary, Φ. This begs the question: how do we obtain an optimized Φ that allows reasonable LVQ results? In this section, the topic of *learning* vector quantization is discussed followed by how this can be accomplished in a crossbar array of resistive switches.

First, we adopt a winner-take-all learning strategy. For each training sample, we choose the closest dictionary element and update only its receptive field. All other neurons' receptive fields remain unchanged. Oja's learning rule [55] is used to learn features from the input training data and build a dictionary, Φ. The rule modifies a neuron's receptive field, $\vec{\phi}$, according to the following equation:

$$\overrightarrow{\phi_{n+1}} = \overrightarrow{\phi_n} + \eta y_n(\overrightarrow{p_n} - y_n\,\vec{\phi}_n) \tag{3.3}$$

where y_n is the neuron activation, given by $\overrightarrow{p_n} \cdot \overrightarrow{\phi_n}$

and η is a learning rate parameter less than one.

A consequence of Oja's rule is that each of the dictionary elements will converge to having a euclidean unit norm. As mentioned earlier, all dictionary elements having the same norm is a prerequisite for our vector quantization algorithm. To provide further motivation for the use of Oja's rule and to understand how it results in normalized weight vectors, consider the following derivation. We begin with Hebb's rule for synaptic weights and add the additional constraint that after each weight update, the norm remain equal to one:

$$y = \overrightarrow{p} \cdot \vec{\phi}^t \tag{3.4a}$$

$$\widehat{\vec{\phi}^{t+1}} = \vec{\phi}^t + \beta y \times \vec{p} \tag{3.4b}$$

$$\vec{\phi}^{t+1} = \frac{\widehat{\vec{\phi}^{t+1}}}{\left\|\widehat{\vec{\phi}^{t+1}}\right\|} \tag{3.5}$$

Equation 3.4 is the standard Hebbian learning rule which can be summarized as "neurons that fire together, wire together." The superscript, *t*, in the equations indicates the state of the weights at each training sample. If *y* is the activity of an output neuron, then the Hebbian rule will increase the weights, $\vec{\phi}$, in proportion to the activity and the strength of the inputs as seen in (3.4). In this equation, β, is simply a learning parameter less than one to provide a gradual weight change across all training inputs, $\vec{p}$. Equation

3.5 imposes the additional constraint that the weights remain normalized to one. This is both intuitive and necessary because application of Hebb's rule without such a constraint would lead to unbounded weight increase; as the weights increase, input and output neurons become more highly correlated which leads to yet more weight increase. This is clearly not physical and so the normalization constraint must be included.

To see how Hebb's rule, combined with the normalization constraint leads to Oja's rule, consider the following. First, assume that the weights are already normalized (in practice this need not be true, so long as the norm of the weights is close to one; how close determines the region of convergence for the learning algorithm and will depend on β).

Assume $\left\|\vec{\phi}^{t}\right\| = 1$

Consider the Taylor expansion of $\vec{\phi}^{t+1}$:

$$\left\|\widehat{\vec{\phi}^{t+1}}\right\|^{2} = 1 + 2\beta y^{2} + \mathcal{O}(\beta^{2}) \tag{3.6a}$$

$$\left\|\widehat{\vec{\phi}^{t+1}}\right\|^{-1} \approx 1 - \beta y^{2} + \mathcal{O}(\beta^{2}) \tag{3.6b}$$

$$\vec{\phi}^{t+1} \approx (\vec{\phi}^{t} + \beta\vec{p}y)(1 - \beta y^{2}) \tag{3.6c}$$

$$\vec{\phi}^{t+1} \approx \vec{\phi}^{t} + \beta\left(\vec{p} - \vec{\phi}^{t}y\right)y + \mathcal{O}(\beta^{2}) \tag{3.6d}$$

$$\vec{\phi}^{t+1} \approx \vec{\phi}^{t} + \beta\left(\vec{p} - \vec{\phi}^{t}y\right)y \tag{3.6e}$$

By starting with Hebb's rule and imposing a normalization constraint, we can use a Taylor expansion to derive Oja's rule. As can be seen in (3.6) this derivation depends on neglecting higher order terms $\mathcal{O}(\beta^{2})$. Thus, as long as β (and thus β^{2}) is kept small, Oja's rule will result in correlated input and output neurons (via Hebb's rule), yet cause

the weights to converge to a Euclidean norm of one, which is useful in many learning algorithms.

This approach is similar to "neural gas" [56] or a K-means algorithm, whereby dictionary elements move to cluster centers within the input data. As will be shown in the next section, the dictionary elements find "features" that are common to all training samples in a cluster.

3.4.1 Crossbar Learning

Each training sample is applied as input to the network as described above. The winning neuron—the neuron with the most similar receptive field—is chosen, and it's activation, *y*, is recorded. Next, a pulse, with duration proportional to *y*, is applied to the network in the reverse direction, collecting the charge on the rows. The collected charge is simply the receptive field of the winning neuron scaled by *y*. This term is then subtracted from the original input to obtain the term $(\vec{p} - y\vec{\phi})$, then multiplied with *y*. A writing voltage, with duration proportional to the final expression, $y(\vec{p} - y\vec{\phi})$, is used as input to the network while grounding the winning neuron. All other neurons are subjected to a half-voltage protection scheme to prevent altering their receptive fields. The choice of writing voltage essentially determines the η term; higher voltages will cause the dictionary to learn faster, but may lead to overshoot, overfitting, and slow convergence. The voltage can also be adjusted as training progresses as in simulated annealing.

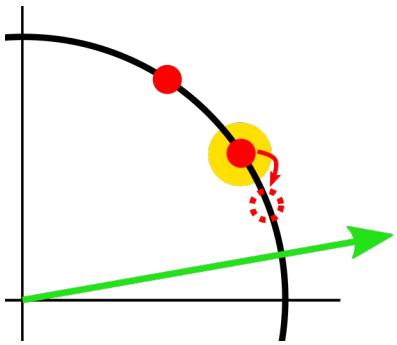


Figure 3.3: Updating a dictionary element (highlighted in yellow) based on a winner-take-all Oja's rule. After sufficient training, all dictionary elements will be constrained to have unit norm (thus lying on the unit hypersphere).

3.5 MNIST Learning

To demonstrate the feasibility of learning in the vector quantization algorithm as implemented in resistive switching hardware, a vector quantization learning module was developed in the simulation framework described above. The MNIST handwritten digit database [57] was used to test the usefulness of the algorithm. In order to perform classification, an additional feature was added to each training input and the crossbar array: the training label for each sample was used to set one of the first 10 pixels in its respective training input. For example, if the training label is a 6 then the sixth pixel (counting from 0) in the first row is set fully on, while the remaining 10 pixels are set identically to 0. Examples of training inputs with embedded labels are shown in Figure 3.4. This label will then be learned, along with the features from the handwritten digit, and later used to classify the test inputs. Learning is therefore conducted in an unsupervised manner—the label is not used to dictate which neuron should win. During inference, the final classification can be performed using a conventional, supervised network or, as was done in this study, simply by taking the strongest label of the winning neuron.

Figure 3.4: MNIST training sample inputs with embedded labels.

After sufficient training, the network forms prototypes of the digits which can be used for pattern matching later. The dictionary elements developed from training are shown in Figure 3.5. The inset shows a larger version of one of the learned dictionary elements. By inspection, we can see that this neuron's receptive field resembles a '5' and that the 5^{th} pixel in the first row is white while the others a dim. This neuron, therefore, would have strong activity if the input is a similarly drawn '5' and the label pixel can be used to identify it.

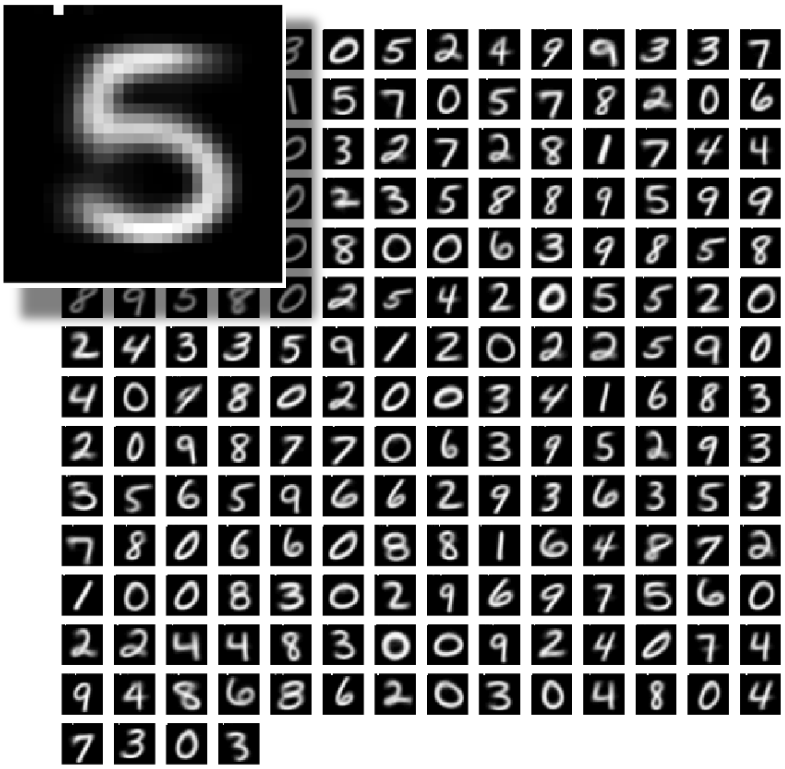


Figure 3.5: Learned receptive fields using a winner-take-all strategy combined with Oja's rule. An enlarged image of one of the fields is shown in the inset.

3.6 Recognition

In order to test the effectiveness of the algorithm, the network was used to classify handwritten digits from the MNIST test database. Inputs to the network appear much like those in Figure 3, but without the labels. The classification procedure is straightforward: an input test image is applied to the network and the neuron with the highest membrane is selected as the winner. The classification label is chosen as index of the pixel (of the first 10 pixels) with the highest intensity. For example, if the neuron with the receptive field shown in the inset of Figure 4 was chosen as the winner, the label would be chosen as 5, since the 5^{th} pixel (counting from 0) has the brightest intensity. This label is then compared to the true label for the test input. The process is repeated for all 10,000 test

samples and the accuracy recorded. The results for different network sizes are shown below in Figure 3.6.

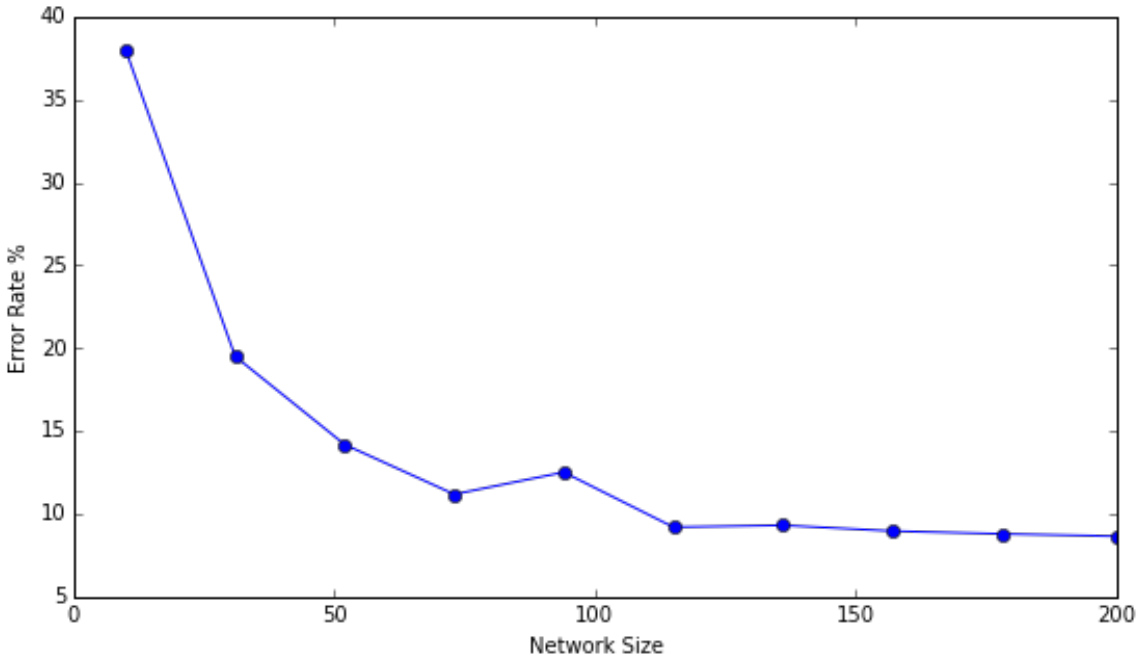


Figure 3.6: Recognition error rate for MNIST test set.

As can be seen in Figure 3.6, larger networks produce more accurate results. This can be intuitively explained by the fact that a larger network can more effectively cover the range of possible inputs; neurons, for example, are able to specialize more towards specific styles of handwriting. While the network results do not demonstrate state-of-the-art performance on the MNIST task, they do suggest a method of performing a common machine learning task in hardware. It should be noted that this naïve approach could be implemented with far fewer resources (area and power) than a conventional software approach. Quantifying the power and area advantages is an area of ongoing research.

3.7 Conclusion

The chapter demonstrated how memristor arrays can be used the accelerate the computations necessary to implement a classic machine learning algorithm, learning vector quantization. A neuromorphic computing approach to matrix-vector multiplication with crossbar arrays was introduced. The technique was used to enable in-memory computing in an analog manner. A winner-take-all approach was combined with Oja's rule to enable efficient in-place, incremental weight updates without the need for a normalization step. The approaches to computation and learning developed in this chapter are applicable to many learning algorithms and will be used again in Chapter 4.

Chapter 4. Sparse Coding

4.1 Introduction

The problem of sparse coding involves finding an efficient representation for an input signal. Given an input vector and dictionary of feature vectors, the goal is to represent the input as a linear combination of features, while using as few features from the dictionary as possible.

Formally, the problem of sparse coding is given as minimizing an energy function:

$$\min_{a,\Phi}(\|\Phi a^T - p\|_2^2 + \lambda\|a\|_0) \tag{4.1}$$

where

Φ is a dictionary with each column representing a dictionary element,

p is a column vector of input signals,

a is a sparse row vector of coefficients,

$\|\cdot\|_2$ is the L_2 or Euclidean norm,

$\|\cdot\|_0$ is the L_0 norm, which is a count of the number of non-zero elements,

λ is a constant chosen to control the sparsity.

$\Phi\vec{a}^T$ is the linear combination of dictionary elements, or the reconstructed input. Thus, the first term in the energy function ($\|\Phi a^T - p\|_2^2$) is the reconstruction error, or how well the sparse encoding represents the original input. The second term ($\lambda\|a\|_0$) is a measure of the sparsity of the encoding. The relative importance of reconstruction fidelity to sparsity is controlled by λ; higher values of λ result in more sparse solutions and in general a poorer reconstruction of the input, while lower values of λ are better able to represent the input, but with a less compact encoding.

Sparse coding is a non-convex problem which can make finding the globally optimum solution difficult and computationally expensive[58]. It is for this reason that a memristive accelerator solution has been investigated. The problem can be separated into two parts in a technique known as forward-backward splitting [59]. The parts can be described as follows: the inference process—given a dictionary, finding an optimal representation of inputs using a combination of dictionary elements [60]–[63]; and the learning process—optimizing the dictionary to improve sparseness and reduce representation error [64]–[66]. We will develop an algorithm, using memristors, and apply it to both simple bar patterns and natural images with the goal of identifying primitive features present in the image. The efficacy of the algorithm is determined by comparing the original image with a reconstruction of the image obtained by linearly combining the extracted features in proportion to their extracted amplitudes.

4.2 Applications

Sparse coding finds a number of applications including data compression, signal restoration, and machine learning [67]–[69]. Because the coefficient vector, $\vec{a}$, of a sparsely encoded signal is primarily filled with zeros, it is sufficient to know only the

indices and values of the non-zero coefficients to (approximately) reconstruct the original signal. If the original signal vector contains m b-bit values, it requires $m \times b$ bits to represent. If it can be encoded using l elements from a dictionary of size n using d-bit coefficients, it requires $l \times (\log_2(n) + d)$ bits. As an example, if $d = b = 8$ and $n = 5m = 5 \times 64$ for (5× overcompleteness), it is not uncommon for $l = .01n$ for natural images, thus the compressed signal requires just $l \times (\log_2(n) + d)/mb \approx 1/10$ of the storage space. For bandwidth limited communication or compressed storage, this technique can be useful. This sparse representation can also be used in systems that employ event-address communication, such as spiking neural network simulators.

Additionally, the sparse coding process identifies the primary features of an input and drastically reduces the representation dimensionality, making it easier for subsequent data analysis. Sparse feature representation has been successfully employed in conjunction with other machine learning techniques to perform object classification [69]–[71].

4.3 Locally Competitive Algorithm

While there are many algorithms that can be used to attack the problem of sparse coding, the Locally Competitive Algorithm (LCA) [72] by Rozell was chosen for its excellent match with the memristor crossbar hardware. LCA falls into a general class known as iterative shrinkage threshold algorithms (ISTA) [73]–[75] that can be used to solve the constrained optimization problem of sparse coding.

While a full description of the locally competitive algorithm can be found in [61], a brief introduction is provided here for discussion. LCA uses a vector of signal inputs

(image pixels in this study) to excite the network. In our approach the pixel values (e.g. intensity in a gray-scale image) are translated to voltage pulse durations with a fixed voltage amplitude, so that the total charge passed by the memristors is linearly proportional to the input, weighted by the memristor conductance. For each output neuron, the crossbar modulates the inputs with a synaptic weight vector (represented by the conductances of the memristors in the same column) and converts them into currents that flow into the neuron. The current is then integrated to determine the neuron's membrane potential, as shown in Fig. 4.1. Additionally, the membrane potential is affected by a leakage term, as well as inhibiting inputs from other active neurons. In LCA, the inhibition is proportional to the similarity of the neurons' receptive fields; this is done to improve sparsity by preventing duplicate neurons with the similar receptive fields from firing. Finally, the membrane potential of a neuron is compared to a threshold to determine the output activity of the neuron. Equation (4.2) describes this dynamical process.

$$\frac{du}{dt} = \frac{1}{\tau}(-u + p^T \cdot \Phi - a \cdot (\Phi^\top \Phi - I)\,) \tag{4.2a}$$

which can be rewritten as:

$$\frac{du}{dt} = \frac{1}{\tau}(-u + (p^\top - a\Phi^\top) \cdot \Phi + a) \tag{4.2b}$$

$$a = T(u,\lambda) = \begin{cases} u & \text{if } |u| \geq \lambda \\ 4u - 3\lambda & \text{if } .75\lambda < |u| < \lambda \\ 0 & \text{if } |u| \leq 0.75\,\lambda \end{cases} \tag{4.2c}$$

where u is the neuron's membrane potential, x is then input vector, Φ is the matrix of the receptive fields (represented by memristor conductances along the columns), I is the identity matrix and a represents the activities of the neurons and is determined by $T(u, \lambda)$, an element-wise thresholding function. The set of active neurons (the non-zero elements of a) forms the sparse representation and their receptive fields contribute to the reconstruction of the input: $\hat{p} = \Phi \cdot a^T = (a \cdot \Phi^\top)^\top$.

From Eqn. (4.2a) it can be seen that the membrane potential dynamics include three terms: a leak term (-u), a driving term proportional to the similarity between the input and the neuron's receptive field ($p^T \cdot \Phi$), and an inhibition term from other active neurons ($-a \cdot (\Phi^T \Phi - I)$). The amount of inhibition is determined by the degree of similarity between the competing neurons' receptive fields, represented by the $\Phi^T \Phi$ term, multiplied by their activation, a, so that only active neurons inhibit other neurons. The $-I$ term is included so that a neuron does not inhibit itself. Equation (4.2b) describes the same system dynamics, but is presented in a form that is easily implemented in a crossbar architecture. In particular, $r = p - \Phi \cdot a^T$ can be considered as an "error" or "residual" term that is fed back to the network [61], [76]. This interpretation is summarized in (4.3) below.

$$\frac{du}{dt} = \frac{1}{\tau}(-u + (p^\top - a \cdot \Phi^\top) \cdot \Phi + a) \tag{4.3a}$$

$$= \frac{1}{\tau}(-u + (p - \hat{p})^T \cdot \Phi + a) \tag{4.3b}$$

$$= \frac{1}{\tau}(-u + r^T \cdot \Phi + a) \tag{4.3c}$$

4.4 LCA Memristor Crossbar Implementation

To implement LCA in memristive hardware, a crossbar array structure is used in which vector-matrix multiplication as well as matrix transpose can be performed simply through read operations. Inputs are interpreted as the duration of fixed-amplitude voltage pulses and applied to the rows of the array. The voltage amplitude is selected to be high enough to be able to perform the computation, but not so high that the memristor's state is altered. "Leaky integrate" neurons are placed on the columns, and the memristors in a given column serve as the synapses for that column's neuron and their weights form the receptive field for the given neuron. This is shown schematically in Fig. 4.1.

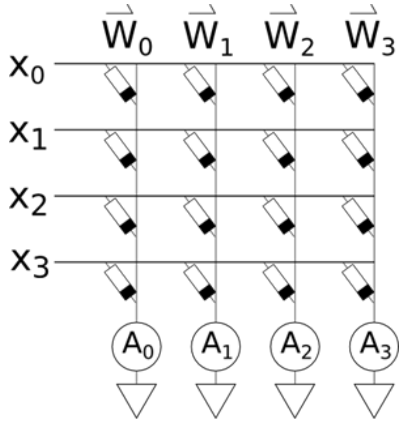


Figure 4.1: Memristor crossbar architecture. Inputs are indicated on the rows as X_x, while the charge is collected on the columns, schematically shown as A_x. Memristors are formed at the crosspoints, and each column of synaptic weights (represented by memristor conductances), W_x, constitutes the receptive field of a neuron.

The integration of the membrane potential is broken into steps.

- First, the residual, *R*, is initialized as the input image, *p*.

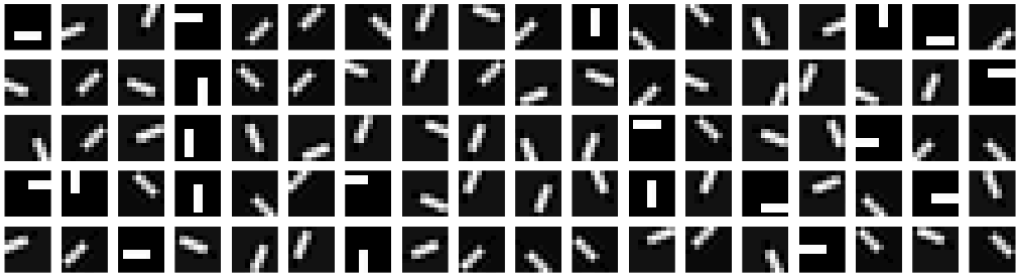
- Next, variable length input pulses, proportional to R, are applied and the charge, C_1, is collected at the bottom. αC_1 is added to the neuron's membrane potential, u. (α is a scale factor to relate charge with pixel intensity).
- A leakage term, proportional to u, is subtracted to obtain an adjusted membrane potential.
- The membrane potential of each neuron is stored and compared to a threshold, and the activation a, is determined, with a fraction being added back to u
- Pulses with durations proportional to a are applied at the bottom of the array, and the charge, C_2, is collected at the left-end of the rows
- The residual is updated as the original input minus the active neuron contribution, $r = p - \alpha C_2$.

The process is repeated in a tic-toc manner, alternating input between the rows and columns, until the membrane potentials reach steady state. In this way, the network is allowed to settle to a state where the set of active neurons is unchanging and a sparse representation that minimizes a cost function that includes of reconstruction error and sparsity is achieved [61]. For a more analog approach the collection of current and leakage adjustment can be performed with a capacitor and low conductance resistor, respectively.

To test the ability of the algorithm to correctly identify underlying constituent components, a dictionary of white bars, with varying placement and angle, on a black background was created; shown in Fig 4.2(a). A test image was then generated by randomly sampling 10 elements from the dictionary and averaging together the selected dictionary elements as shown in Fig 3(b). The sparse coding LCA with $\lambda = .02$ was run

on the test image to determine if the indices of the active neurons matched the dictionary elements used to construct the image.

(a)



(b)

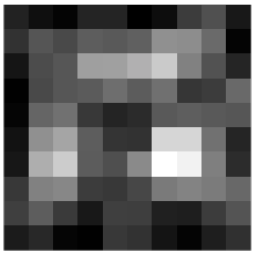


Figure 4.2: Standard bars test for inference. (a) Test dictionary of white bars on black background—random sample of 90 out of 392 dictionary elements shown. (b) Example test image constructed by averaging 10 randomly selected dictionary elements.

The process was repeated on 1000 test images and it was found that the sparse code exactly matched the components used to construct the image with a success rate of 94%. This demonstrates the ability of LCA sparse coding to discern constituent features from a composite image. On more complex inputs, natural images for example, the goal is to extract the features that make up the image. This information can then be used to reconstruct the image or even identify objects within it.

4.5 Dictionary Learning

The LCA algorithm, as described in [61], addresses the first challenge of feature extraction—namely, how to sparsely represent an input with a given dictionary, but LCA

itself does not cover the topic of how to learn a dictionary of feature primitives (ideally in an unsupervised fashion) while taking into consideration the efficiency and limitations imposed by the hardware. In this section we discuss our approaches to learn the dictionary of feature primitives using memristor crossbars.

For optimal sparse coding, it is necessary to find a dictionary that is well suited to the types of inputs to be sparsely encoded. The extraction of features from an input is only possible if the dictionary contains those features. We first review the use of stochastic gradient descent and discuss the drawbacks of implementing this algorithm in hardware. This discussion will motivate the use of the learning technique (WTA + Oja's rule) developed in Chapter 3 to learn dictionaries of features from training samples; the learned dictionary will resemble the feature primitives found in the training set. The first experiment will use horizontal and vertical bars to test the ability of the learning algorithm to identify features. Next, the same algorithm will be applied to samples drawn from natural images; the learned dictionary elements resemble Gabor filters which are theorized to underlie the receptive fields in the visual cortex [65].

4.5.1 Stochastic Gradient Descent

An intuitive approach is to begin with an untrained dictionary of random receptive fields, apply the LCA sparsification algorithm to a training patch to obtain the coefficients, and then use stochastic gradient descent to adapt the receptive fields of active elements to reduce the reconstruction error [65], [77]. Briefly, the stochastic gradient descent algorithm modifies the weight vectors of active neurons by an amount proportional to the negative of gradient of the cost function's gradient for each training sample. It is a first-order optimization approach commonly used in machine learning, but

requires the computation of the error gradient for each training sample.

At each step, a training sample is chosen at random and applied to the network. The network is then allowed to evolve following the LCA algorithm discussed in Section 4.3. For training, λ was arbitrarily chosen to be 0.3 as this produces an intermediately sparse solution. After the network has reached steady state, only a few neurons will be active. The gradient of the error with respect to the dictionary is given as $\nabla E = -(p - \Phi a^T) \otimes a$. To descend the error field, the following learning rule is applied: $\Delta\Phi^\top = \beta(p - \Phi a^T) \otimes a$ where β is a scale factor << 1; a full derivation is given in [65].

While gradient descent in conjunction with LCA produces very good results, it is a complex learning process, requires normalization of the weight vectors, and results in negative as well as positive weights—implying a non-physical negative conductance in the memristor. The error produced by LCA is typically small, and thus the adjustments made to the receptive fields are also small. Further, for each training step, the full LCA algorithm must be run until steady state to obtain the representation coefficients, and then training pulses must be applied for each active neuron. The result is a network that takes significant resources and time to train effectively.

4.5.2 Winner Take All & Oja's Rule

As an alternative to the complexity of stochastic gradient descent, we opt for the same training algorithm developed in the previous chapter, namely, a winner-take-all strategy coupled with Oja's update rule. To test the ability of the learning algorithm to learn a basis dictionary, we performed the following experiment:

- An array of 10 vertical and 10 horizontal lines, each 1 pixel wide, shifted and embedded in a 10x10 black pixel field was generated. The elements are shown Fig. 4.3.
- From this array 2 elements were chosen and linearly combined; this combination was added to the training set. Thus the training set consisted of all $\binom{20}{2} = 190$ combinations of 2 elements.
- The network was initialized to have 20 neurons with random receptive fields.
- The WTA/Oja learning was performed as described in Chapter 3 repeatedly using the training set as inputs.

(a)

(b)

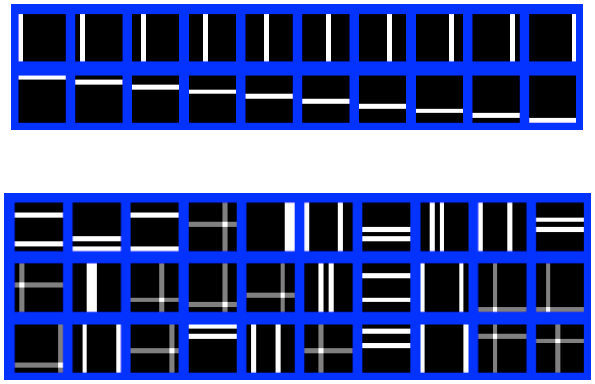


Figure 4.3: (a) Basis set of vertical and horizontal elements used to form the training samples. (b) A random sampling of 30 out of 190 training samples formed by selecting 2 of the bases from (a).

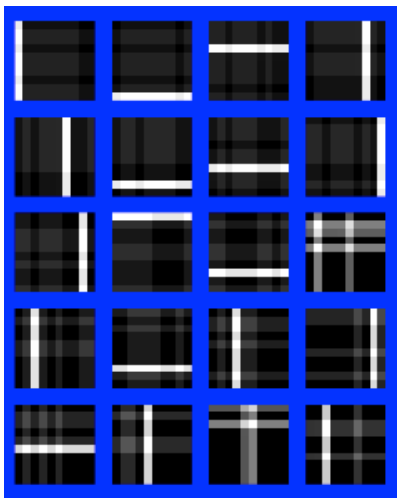


Figure 4.4: Learned dictionary from training patterns consisting of 2 basis elements.

As can be seen in Fig. 4.4, most dictionary elements have evolved to resemble a single vertical or horizontal bar. This demonstrates that the learning algorithm is able to discern the basis set that was used to construct the training examples. The results are encouraging and suggest that the training algorithm could also find a basis set for natural images.

4.5.3 Application to Natural Images

The winner-take-all—Oja learning rule was then applied to the more complex task of learning a dictionary from a set of natural images. To train the network, 10x10 pixel patches are sampled from a set of 9 natural images shown in Fig. 4.5. Each patch is scaled and reshaped into a column vector as input. Training then proceeds as described in Chapter 3, Section 3.4.1.

Figure 4.5: Natural training images. All 110,889 10x10 pixel patches are extracted (by sliding the 10x10 sample window across and down each of the 9 120x120px input images) for training.

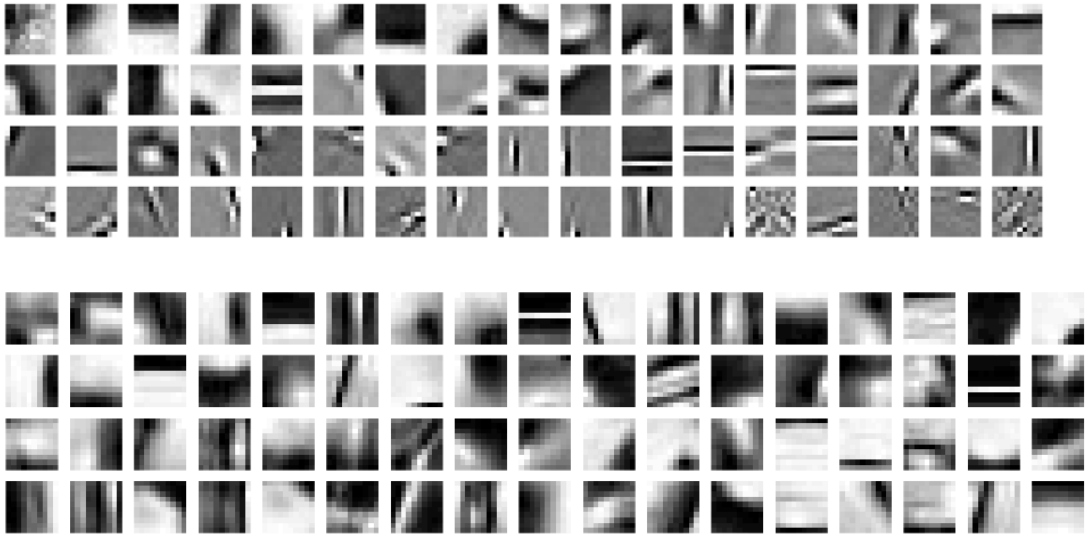


Figure 4.6: Receptive fields formed using (a) stochastic gradient descent and (b) WTA and Oja's rule directly (neglecting device model) on natural images.

A comparison between the receptive fields obtained by gradient descent and the WTA-Oja learning rule is shown in Figure 4.6. The two sets of fields are qualitatively similar (quantitative LCA results are compared in a later section) and resemble gradients and Gabor filters of various orientation and frequency. These fields are similar to those

found by other techniques and thought to exist in the mammalian visual cortex [62]. Gabor filters are useful for identifying edges and identifying textures [78], [79], so the emergence of Gabor-like receptive fields from a Hebbian-derived learning rule is a promising outcome.

4.6 Sparse Reconstructions

To test the learned dictionaries, a natural image, shown in Fig. 4.7 and not included in the training set, was compressed using LCA and the trained memristor network and then reconstructed from the active coefficients obtained from compression and their corresponding dictionary elements.

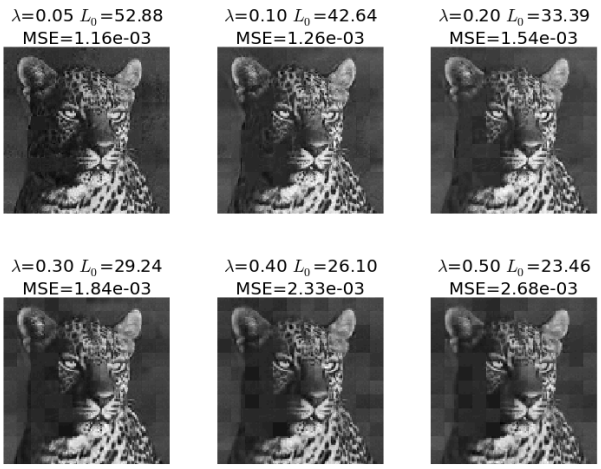
Figure 4.7: Test image of a leopard. 10x10 non-overlapping pixel patches were extracted from this 120x120 resolution image to test the performance of the training algorithms in conjunction with LCA used in this paper.

During this phase, the test image is broken into 144 square patches of 10x10 pixels each. Each 100-valued input is applied to the memristor network, consisting of 300 columns, where each column consists of a 100-valued dictionary element (i.e. 100 rows). The network dynamics are allowed to reach steady-state following LCA. The activities of the neurons are recorded and this forms the sparse representation of the image (since

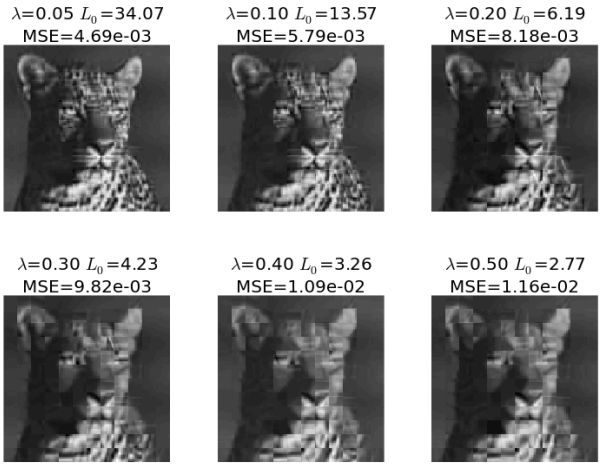
most of the activities are exactly 0, only the index and value of the above threshold neurons are recorded). The number of active coefficients (equivalently, the compression ratio) is influenced by the λ parameter in the thresholding function of (Eqn. 4.2c).

Next, the set of patches are reconstructed by linearly summing the receptive fields of the active dictionary elements, weighted by the stored activities obtained during sparse coding. These decompressed patches are then tiled to form the reconstructed image; examples of reconstructed images are shown in Fig. 4.8. By increasing the threshold parameter λ, it becomes harder for neurons to become active and contribute to the coded representation, and thus the solution becomes more sparse. For each reconstructed image in Fig. 4.8, the λ parameter used during LCA as well as the resulting number of active neurons (average L_0 over all 144 patches) is shown above it. As λ increases, the solutions become more sparse and the reconstruction quality in general deteriorates, so LCA sparse coding can be thought of as a form of lossy compression.

(a)



(b)



(c)

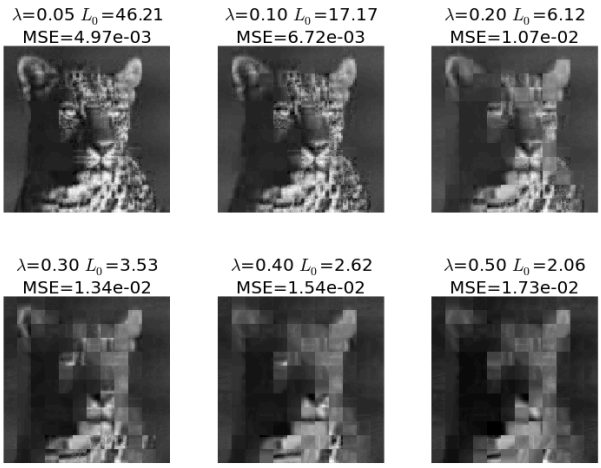


Figure 4.8: Image reconstructions using coefficients obtained from the LCA algorithm after different training implementations: (a) gradient descent, (b) WTA software implementation, (c) WTA using the device model.

While Fig. 4.8 (a-c) all use LCA for reconstruction, they differ in the method used to train the dictionary used by LCA. The dictionary trained with stochastic gradient descent produces a good, low-distortion reconstruction of the image, particularly at low-sparsity solutions (with an average of 42.6 active neurons per patch). At a high $\lambda = 2.06$, however, the gradient descent result appears worse than that obtained by the WTA

software implementation. This can be understood on the grounds that gradient descent was trained and optimized at λ=0.3, while the high sparsity solutions requires a higher λ. The WTA-Oja approach shown in (b-c) is offered as an alternative to stochastic gradient descent and requires significantly less computation. From Fig. 4.8(b) it can be seen that the use of Oja's rule in a winner-take-all training strategy performs well, with reconstruction quality degrading gracefully with increasing λ. Fig. 4.8(c) shows that WTA-Oja strategy implementation with the device model works with the low-sparsity constraint but exhibits markedly reduced performance as the solution becomes more sparse. This can be attributed to the non-linear response to programming arising from the window function term ($F(w,V)$) of the device model (see Eqn. 4.4b below). This nonlinearity, shown in Fig. 4.11a below, distorts Ojas rule by reducing the effectiveness of programming pulses when the device state is near its extremes (it becomes harder to erase a device that is already close of OFF and harder to program a device that is close to ON); the issue will be addressed in a later section.

4.7 Impact of Device Variability

Memristor devices are inherently variable since the resistance change is driven by the migration of oxygen vacancies (for an oxide-based memristor) which are essentially defects in the oxide matrix. The resistance switching process is strongly affected by the local field and oxygen vacancy profile and shows significant device-to-device and cycle-to-cycle variations [47]. In order to assess the impact of device variability on the network performance, random variations were introduced to the device modeling parameters. The device model is repeated from Chapter 2 below for reference:

$$\frac{dw}{dt} = \eta_1 \sinh(\eta_2 V) F(w, V) - \frac{w}{\tau} \tag{4.4a}$$

$$I = w\,\gamma \sinh(\delta V) + (1 - w)\alpha(1 - \exp(-\beta V)) \tag{4.4b}$$

Percent variation for each parameter was chosen to reflect realistic deviations resulting from processing conditions and material properties; sources of variation for each parameter are discussed below. Variations in these parameters affect not only the computation stage but also the learning stage. In these studies, the decay term, $-\frac{w}{\tau}$, was considered to be negligible and was not included. This was done so that the network learning algorithms become time-independent. This is a reasonable approximation so long as the decay time constant, τ, is sufficiently large compared to the speed at which the LCA algorithm and network learning is performed. Experimental evidence has suggested this is the case. It is noted that if the network weights decay appreciably, any system using these weights will likely have to continue learning periodically to reinforce and restore the weights (interestingly there is evidence that this occurs in mammalian brains as well [80], [81]), although the weight decay effects are an area for future research.

Variable	Nominal Value	Rel. Std. Dev. (%)
η_1	7.21e-8	3
η_2	18.54	1
γ	2.05e-5	10
δ	1.03	2
α	1.03e-5	10
β	0.515	5

Table 1: Nominal device parameters with estimated variances.

4.7.1 Sources of Variation

η_1 describes the ion hopping dynmics and is largely influenced by the attempt frequency and hopping distance of oxygen vacancies and is related to film non-homogeneity.

η_2 characterizes the distortion of the energy barriers for ion hopping in response to an applied electric field. Variation can occur as a result of local field enhancement, though this is mitigated by the averaging effect of having multiple parallel conduction paths form.

γ is a prefactor for the tunneling current associated in transport through the conductive regions. While there are many variables that affect γ, total device area can have a significant impact. Area variations arise primarily from lithographic constraints.

δ appears in the sinh term of the tunneling current. Variations can occur in tunneling distance between the conductive region and the electrodes, though this is somewhat self-regulated by the dynamics of filament growth when a series resistance is present in the circuit.

α is a prefactor for the Schottky current component. Like γ, variations can be attributed primarily to area nonuniformity resulting from lithographic constraints.

β is found as a multiplicative factor with the voltage in the schottky current equation. Variations in β represent variations in the ideality factor of the diode.

4.7.2 Impact

The effects of these variations were tested using a Monte Carlo approach and the resulting device behavior can be seen in Fig. 4.9. Here we model the effects of variation in the current equation variables (Fig. 4.9(a)) and dynamics equation variables (Fig.

4.9(b)) using a normal distribution centered at the nominal value and with relative standard deviations given in Table 1. Figure 4.9(c) shows the combined effect of these variations. The results from the combined variation in Fig. 4.9(c) appear reasonable and are consistent with variation observed in real devices. The magnitude of the effective variation, defined as $(I_{max} - I_{min})/I_{min}$ where I_{min} *and* I_{max} are measured at the highest conductance state (i.e. after 20 programming pulses), can be in excess of 100% and is certainly significant in memristor devices. The large variation can be attributed to uncertainties in fabrication but more fundamentally uncertainties in the local electrochemical environment for ion motion and the stochastic nature of the ion hopping process. To determine the effects of this large variation on the network performance, we repeated the WTA training with this device variation model and the results are shown in Fig. 4.10. The device variations result in less distinct feature elements and consequently poorer image reconstructions.

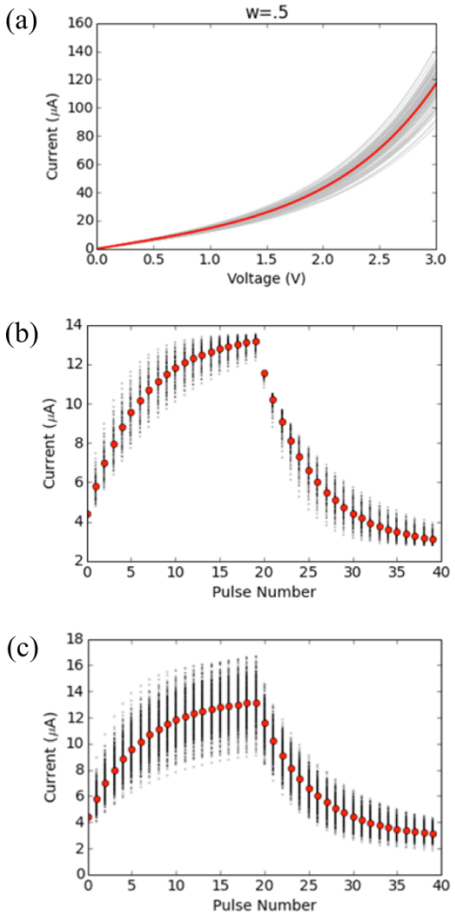


Figure 4.9: Effects of variations in (a) current equation parameters, (b) state dynamic equation parameters, (c) all parameters.

(a)

(b)

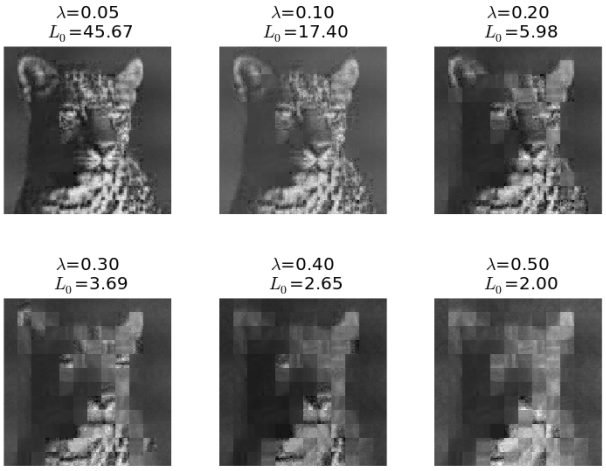


Figure 4.10: (a) Receptive fields resulting from WTA training using a device model that incorporates parameter variations. (b) Resulting image reconstructions using these receptive fields with LCA.

4.8 Nonlinearity Compensation

From the results above, it is evident that, while the LCA algorithm adapts to the dictionary, the impact of the device non-idealities results in degraded performance. The non-idealities can be classified into two effects: parameter variation, as discussed previously, and nonlinearity during training with respect to programming pulse number (seen in Fig. 4.11(a) and resulting from the window function, *F(w,V),* in Eqn. 4.4b). The nonlinearity reduces the effectiveness of Oja's rule since Oja's rule assumes the weight update is linearly proportionally to the neuron activity and error while, in an actual memristor, the weight update is additionally affected by the window function: as the memristor reaches saturation (near the levels of minimum and maximum conductance), it requires increasingly more programming time to achieve the same conductance change. Below we show that effects of nonlinearity in programming can be corrected and that, as

a result, algorithm performance can be greatly improved even in the presence of large device variations.

Using the device model, a relationship between desired state change and pulse duration can be derived to compensate the nonlinearity effect and produce desired linear weight updates. Employing such a relation, once the state update is calculated using Oja's rule, a corresponding pulse duration can be calculated and applied to the memristor. Given a desired state update of Δ and current state of *w*, the pulse duration is given as:

$$f(V)\log\left(\frac{w-k}{\Delta+w-k}\right) \qquad (4.5)$$

where w is the current weight, $f(V)$ depends only on the voltage and material parameters, Δ is the desired change, and $k = \begin{cases} 1 & \text{if } \Delta \geq 0 \\ 0 & \text{if } \Delta < 0 \end{cases}$

As can be seen from Eqn. (4.5), it is necessary that the current state of the memristor be known. This can be obtained by applying a read pulse on the columnar electrode and simultaneously reading out the states of the memristor from each row. This comes at the expense of an additional step in the training process. However, only the synaptic weights associated with the winner neuron needs to be read out, and the read can be performed in parallel which will help minimize delays in the training process. It should be noted that if $w + \Delta > 1$ or $w - \Delta < 0$ then Eqn. (4.5) gives a non-finite pulse duration; in this case, a fixed long pulse is used to drive the *w* close to bound.

Figure 4.11 contrasts the effects of using an adjusted pulse duration with those of using a fixed duration for a series of 10 positive pulses, followed by 10 negative pulses,

repeated 3 times, showing the scheme can effectively compensate the nonlinear effect and produce the desired linear weight updates.

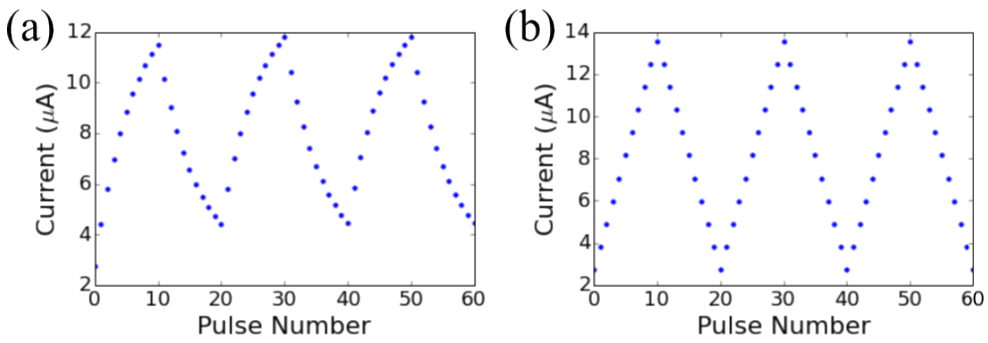
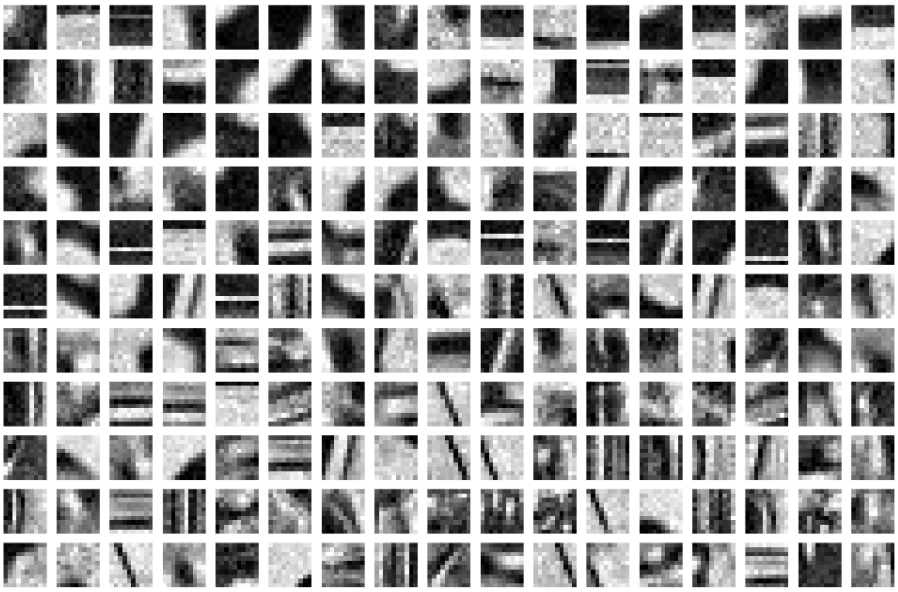


Figure 4.11: Device programming linearization. (a) Device read current following repeated write and erase pulses of fixed duration. (b) Linearized device behavior obtained through compensated write/erase pulse duration in accordance with Eqn. (4.5).

Using the non-linearity compensation scheme developed above, the WTA training was repeated. It can be seen in Fig. 4.12(a) that the receptive fields after training using this method are closer to those obtained using the ideal software model, and importantly, that the Euclidean norm of the fields is closer to one with less deviations. Fig. R(b) demonstrates that the reconstruction performance is indeed greatly improved as a result, even in the presence of large device variations.

(a)



(b)

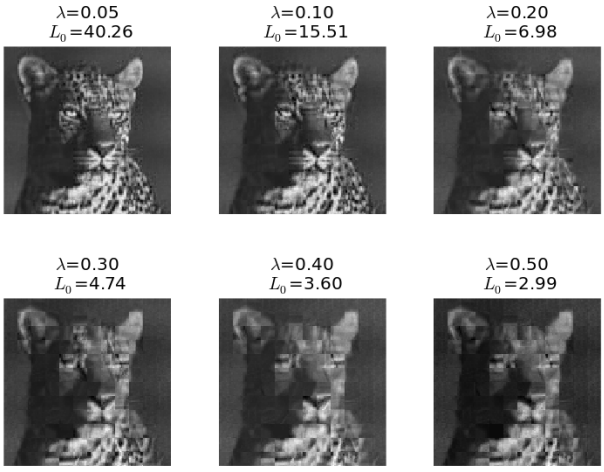


Figure 4.12: Linearization improvement. (a) Receptive fields obtained through non-linearity compensation when using WTA training with a device model that incorporates parameter variations. (b) Improved LCA image reconstructions using the compensated fields.

4.9 Quantitative Error

To quantitatively measure the degree of distortion, the mean-squared error (MSE) between the original image pixels and the reconstruction is calculated:

$$MSE = \frac{1}{100}\sum_{i=1}^{100}(X - \Phi^{\top}a)_i \qquad (4.6)$$

This metric can be used to compare algorithm performance and measure the effect that device parameters have on reconstruction quality as is shown in Figure 4.13. From the figure, it can be seen that while the full LCA training using gradient descent yields the best results at lower sparsity, WTA can yield comparable performance with reduced complexity in implementation and even surpass gradient descent at high sparsity. Further, the result shows that the network is capable of compensating device-to-device parameter variations that result in maximum current levels differing by over 100%. On the other hand, dictionary training suffers from the device non-linearity, but this limitation can be overcome using the training pulse-width compensation scheme discussed the previous section. It is interesting to note that results using the device model with variations, coupled with the compensation scheme outperformed the compensated device model without variations and even the software implementation, though the improvement is slight. It is hypothesized that parameter variations may help prevent the dictionary from becoming trapped in local minima during training, although detailed analysis of the effect of variations (noise) in network performance will be a topic of future research.

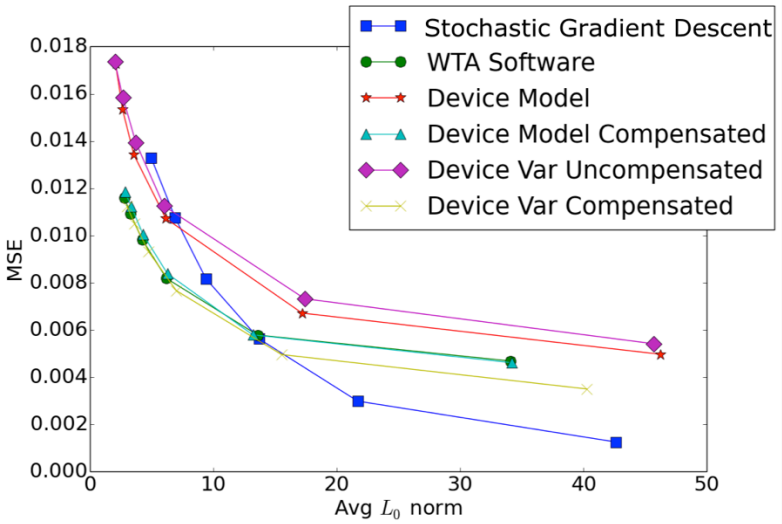


Figure 4.13: Quantitative comparison of LCA reconstructions. Comparison between gradient descent, winner take all, and device models—with and without parameter variations and non-linearity compensation.

4.10 Device Failures

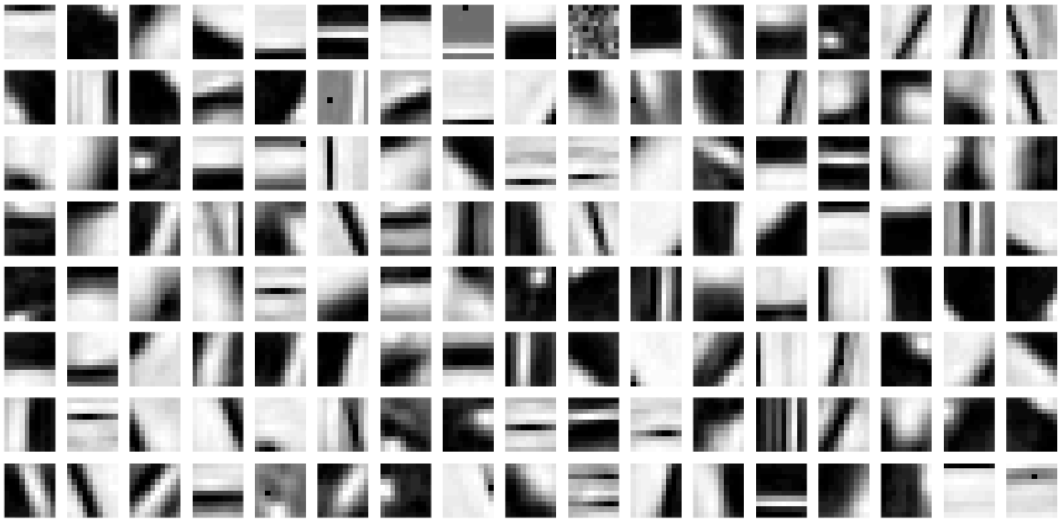
Memristive devices can fail in a number of ways and it is important in some applications that isolated device failures not result in catastrophic system failure. To investigate the effects of device failure on LCA sparse coding performance, we simulated algorithm performance with defective devices.

In this preliminary work, we chose to examine stuck-at faults since these are expected to be of more severe consequence than transient faults. Since learning is an iterated process, transient faults are expected to rectified with additional training. During network inference, transient faults are expected to result in degraded sparse code generation and/or degraded image reconstruction, but the effect should be limited to the duration of the fault. Stuck-at faults, on the other hand present a persistent over- or under-estimation of the true or desired weight within the array and it was found that their presence can distort network training.

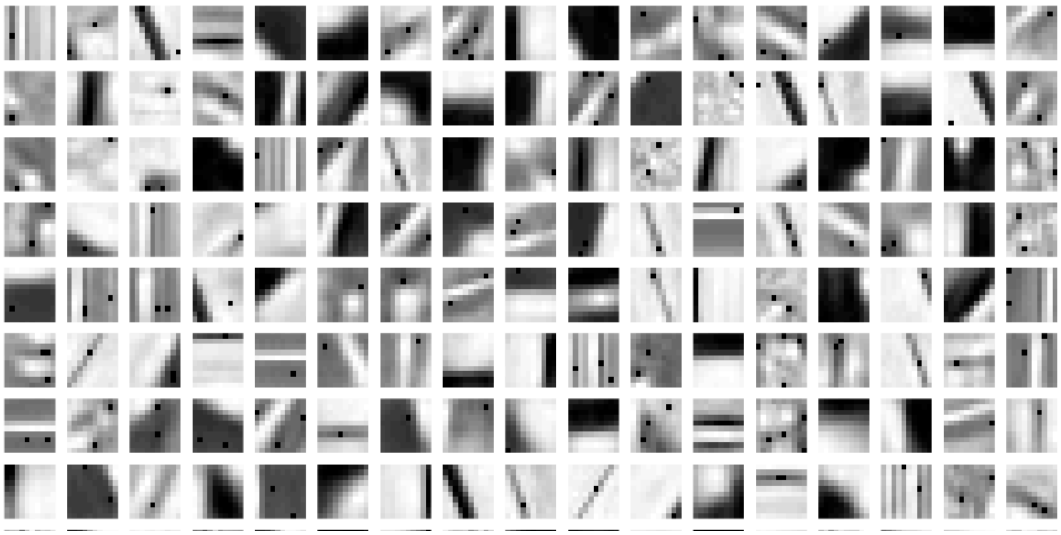
4.10.1 Nonconductive Defects

Devices that fail in the OFF state are said to be stuck-at zero (SA0). This can be a result of a broken electrode or a failure to form a conductive path between the top and bottom electrodes. To test LCA performance, we first generate a memristor array with randomized weight values, following the same procedure for simulating a fault-free array. To simulate SA0 faults, we then generate a second boolean matrix of the same size where each element is True with a probability given by *p*. The locations of True elements are stored and weight values corresponding to these locations are set to 0. At each training step, the learning rule updates for these locations are ignored and the value kept at 0.

(a)



(b)



(c)

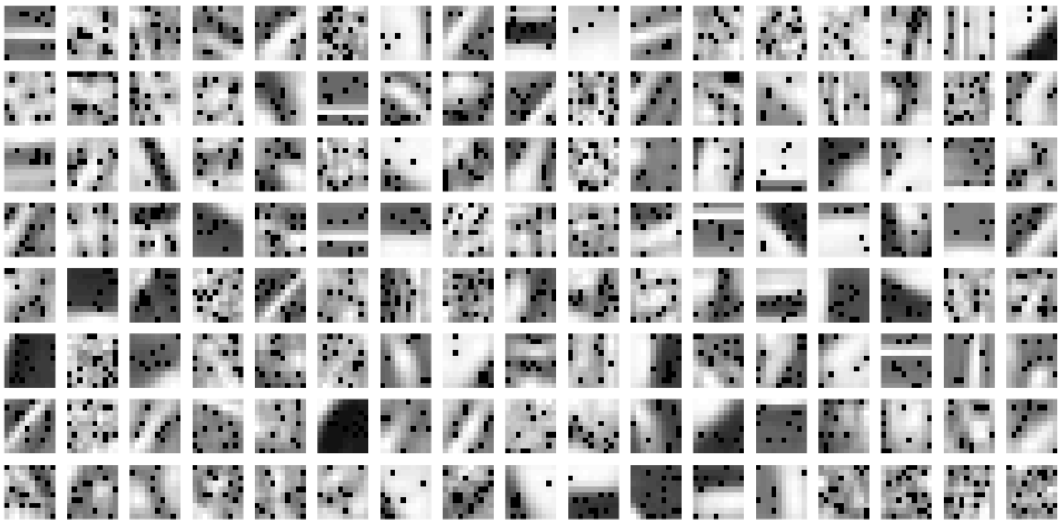


Figure 4.14: Demonstrating the effects of SA0 faults on dictionary learning. (a) 0.1%, (b) 1% and (c) 10% of devices randomly selected as defective.

Figure 4.16 shows the effect of nonconductive faults on algorithm performance. It can be seen that SA0 do not degrade network performance, and in fact may improve sparse coding error slightly. It is believed that this may be the result of SA0 faults further de-correlating dictionary elements, but more investigation is required to confirm if this is the case. At higher rates of SA0 faults (not shown), algorithm performance is observed to be negatively impacted.

4.10.2 Maximally Conductive Defects

Memristors can also fail in such a way that they are effectively stuck in the ON state. These stuck-at one (SA1) faults can be the result of pin-hole defects in the switching medium that occur during fabrication, static discharge, abnormally high programming voltages, or ionizing radiation.

The simulation of SA1 faults follow the same procedure for SA0, with the exception that faulty weights are given a value of 1 rather than 0. It was observed that unlike SA0 faults, SA1 defects negatively impact algorithm performance (Fig. 4.16). Even a single SA1 fault effectively renders the column in which is occurs unusable. This can be understood by examining the effect of Oja-rule training on memristor weights. After training, the weights will converge such that the L_2 norm of weights in a given column will be equal to one. If one (or more) of the weights is stuck at one, the learning rule will force the remaining weights close to zero. The dictionary element is then effectively able to encode only a single pixel. The element can still be used when sparsity constraints are weakly enforced, but at higher sparsity, the algorithm will not use an element that encodes only one pixel. Furthermore, the presence of SA1 faults can effectively prevent training of other dictionary elements as seen in figure below.

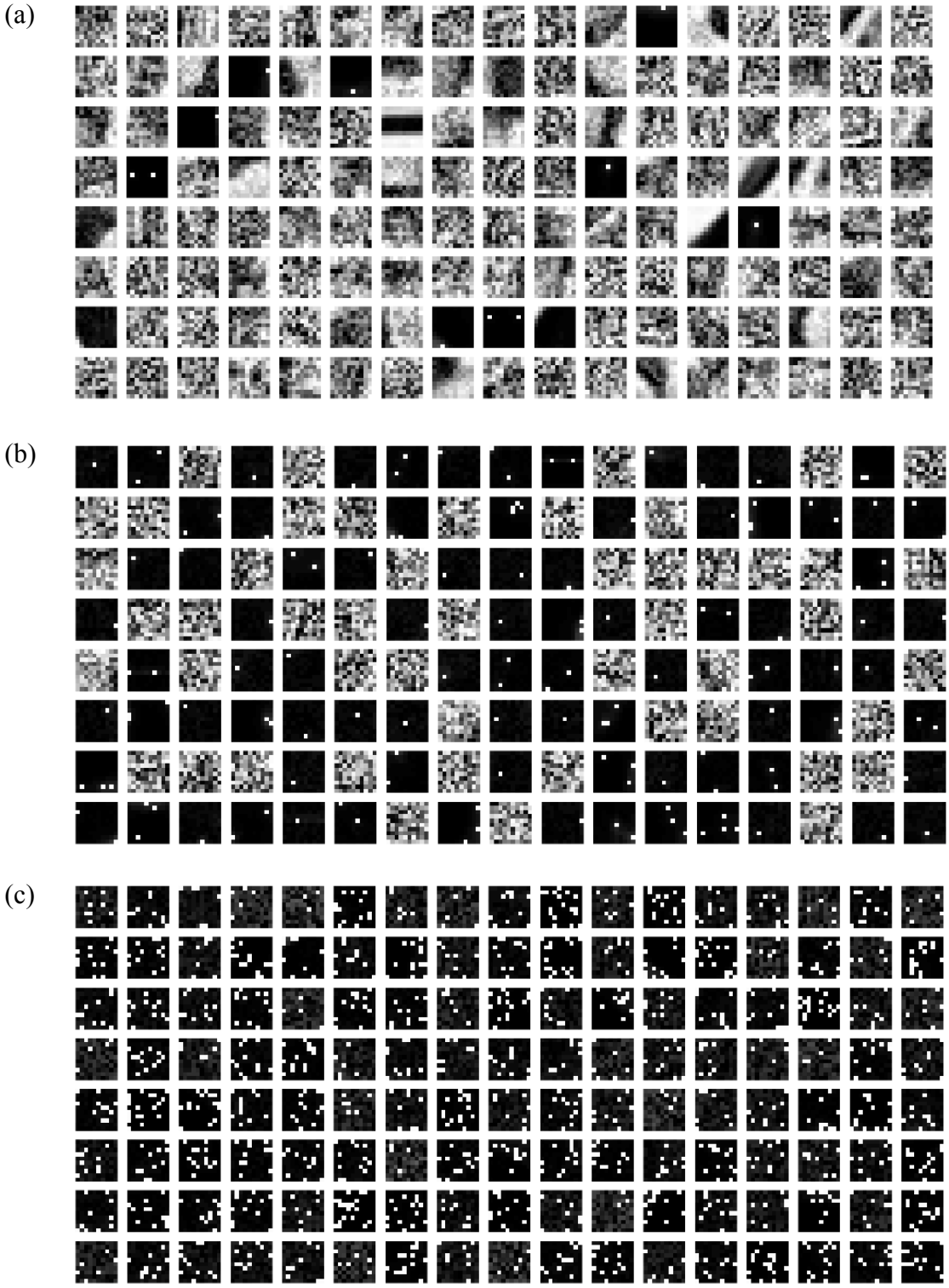
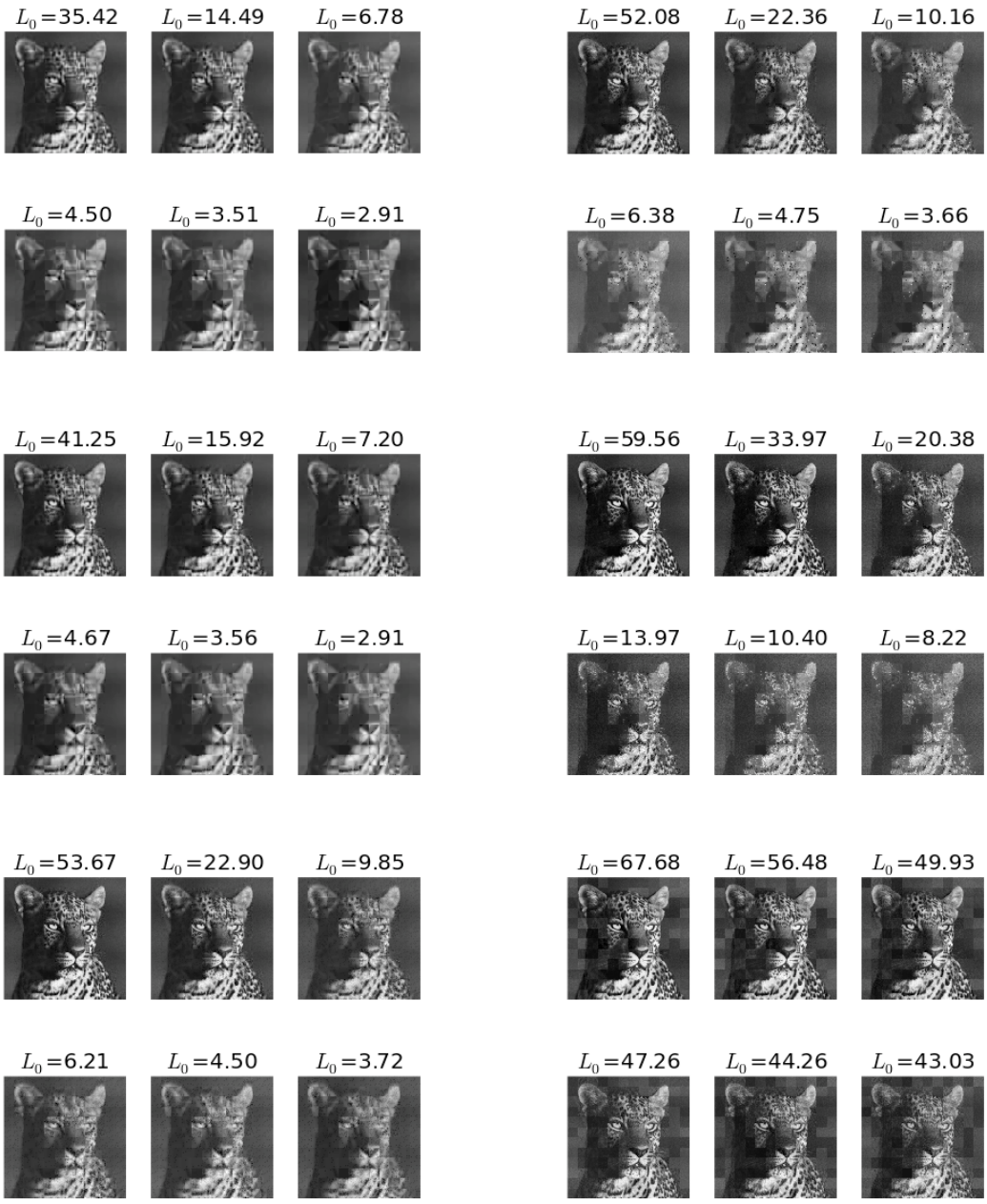


Figure 4.15: Demonstrating the effects of SA1 faults on dictionary learning. (a) 0.1%, (b) 1% and (c) 10% of devices randomly selected as defective.

Even if a dictionary element (crossbar column) does not contain a SA1 fault, its training can be impeded by SA1 faults that are present elsewhere in the array. This results from the winner-take-all approach to training. Before training all memristor weights will start in a relatively low conductance state (with the state naturally settling near close to 0), with the exception of the SA1 faults. When the training samples are applied the columns with SA1 faults will naturally win simply because of their high conductance fault. The persistent winning of the faulty elements effectively prevents the other elements from being trained.


L_0=35.42
L_0=14.49
L_0=6.78
L_0=52.08
L_0=22.36
L_0=10.16
L_0=4.50
L_0=3.51
L_0=2.91
L_0=6.38
L_0=4.75
L_0=3.66
L_0=41.25
L_0=15.92
L_0=7.20
L_0=59.56
L_0=33.97
L_0=20.38
L_0=4.67
L_0=3.56
L_0=2.91
L_0=13.97
L_0=10.40
L_0=8.22
L_0=53.67
L_0=22.90
L_0=9.85
L_0=67.68
L_0=56.48
L_0=49.93
L_0=6.21
L_0=4.50
L_0=3.72
L_0=47.26
L_0=44.26
L_0=43.03

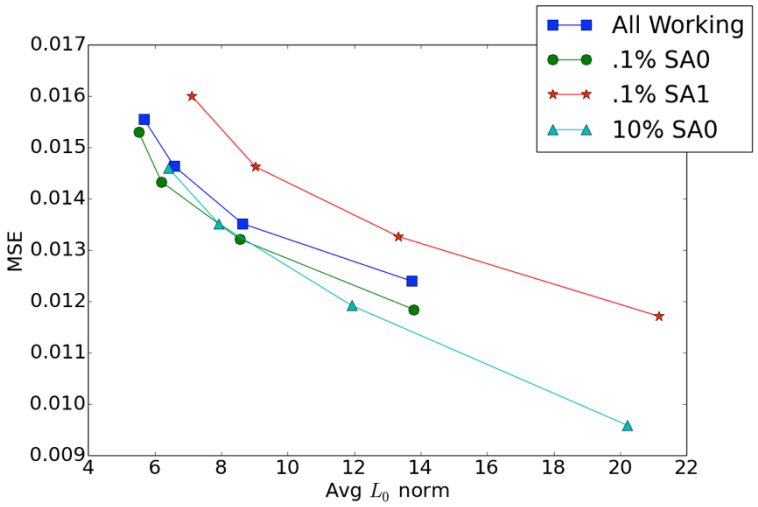


Figure 4.16: Quantitative comparison of LCA with SA faults. The left column shows the results of sparse reconstruction using a SA0 dictionary, while the right columns shows the results of a SA1 dictionary. Defect densities range from .1% (row 1), 1% (row 2), to 10% (row 3).

4.11 Conclusion

In-memory sparse coding using the locally competitive algorithm (LCA) and crossbar arrays of memristors has been presented. The adaptation of the algorithm to memristor hardware was discussed and compared with ideal software solutions. The WO_x device model and simulation framework developed in Chapter 2 is used to simulate realistic array behavior and device non-idealities including parameter variation and failure were considered. Key to the success of the implementation is the reliance on feedback mechanisms to compensate device non-idealities. The iterative forward-backward algorithm proposed to numerically integrate (with a crossbar array) the neuron dynamics (Eqn. (4.2)) can effectively compensate over- and under-represented pixels that may be present in the dictionary features as well as noise. Likewise, the iterative application of Oja's rule can compensate device over- or under-programming while the winner-take-all approach targets untrained dictionary elements (because their norms are

greater than one). However, the effect of the window function limits the effectiveness of weight updates by causing an asymmetry in weight change depending on the current weight. To solve this problem, an adjusted programming pulse scheme is presented and good results are obtained.

Unlike device variability, the effect of SA1 memristors presents a serious challenge for LCA sparse coding. Given that a single SA1 fault in a column renders it useless for sparse coding, as the probability p of faults increases, the dictionary size is effectively reduced. The impact is also related to the patch size; more inputs per patch increases the likelihood that at least one of them is defective. Given a patch size n, the probability that a dictionary element is defective is given by $1-(1-p)^n$. In the examples given in the study, a 16x16 input patch was used; with even a low defect rate of p=.001, there is a 22.6% chance that a given dictionary element is defective. For this reason, smaller patch size may be preferable, though this will reduce the total sparsity of an image encoded patch by patch. Techniques to repair a SA1 fault, or at least remove its current contribution from the column, would prove very useful in developing larger robust systems in the face of these kinds of faults.

Chapter 5. Test and Measurement

Through the course of the research, several test and measurement platforms have been constructed. Resistive switches do not operate without additional circuitry to control reading and writing operations. In order to provide high performance and minimize area utilization, the crossbar arrays can be directly integrated with and over CMOS circuitry. Previous work has demonstrated the feasibility of integrating digital switching devices with CMOS devices [31], [82], but an integrated analog array had not been shown. Presented below is the successful integration of analog switching WO_x memristor arrays on a test and measurement setup developed as part of the DARPA SyNAPSE program.

An additional measurement board was developed to allow testing of larger crossbar arrays of devices. This system included the development of a system-on-chip targeted for Xilinx FPGAs to provide both control for the crossbar array and for executing neuromorphic algorithms. Hardware and software was co-designed to successfully use the arrays to accelerate algorithm execution.

5.1 CMOS Integration

Since the memristors' operation and fabrication do not involve the single-crystalline Si substrate, they can be fabricated over existing circuitry in the Back-end-of-the-line (BEOL) fashion. In this study, the memristors were fabricated on top of an integrated test circuit designed in collaboration with HRL Laboratories. The chip acted essentially as a decoder: given an address for a row and column, connections were made

from the input to the corresponding memristor electrodes. Signals could then be passed to the device for reading or writing the device state. The first chips were designed with IBMs 180nm 7RF process using Al interconnects; later devices used the 90nm 9RF process using Cu interconnects. Schematic of the memristor integration processes and the scanning electron micrographs (SEMs) of the exposed Cu vias (for the 90nm chips) and integrated memristor chips are shown below in Figure 1:

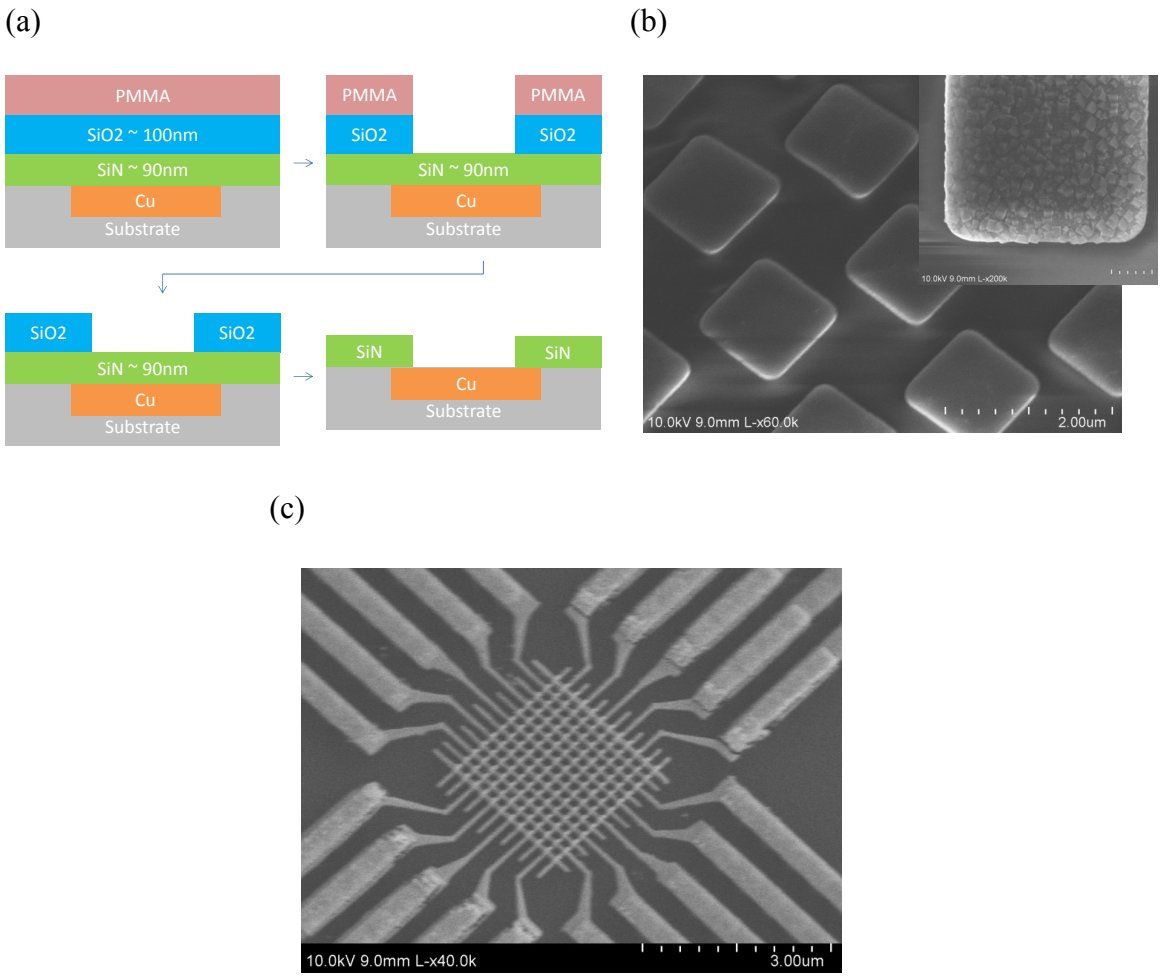


Figure 5.1: CMOS Integration. (a) Fabrication flow for exposing Cu landing pads to electrically connect electrodes. (b) An SEM of the landing pads. (c) An SEM of a completed 10x10 array. Adapted from [83].

In order to provide the necessary control and device signals, a test and measurement setup that included a probe card, a National Instruments data acquisition system, and custom software written in Matlab was used. A schematic of the setup is shown in Figure 2.

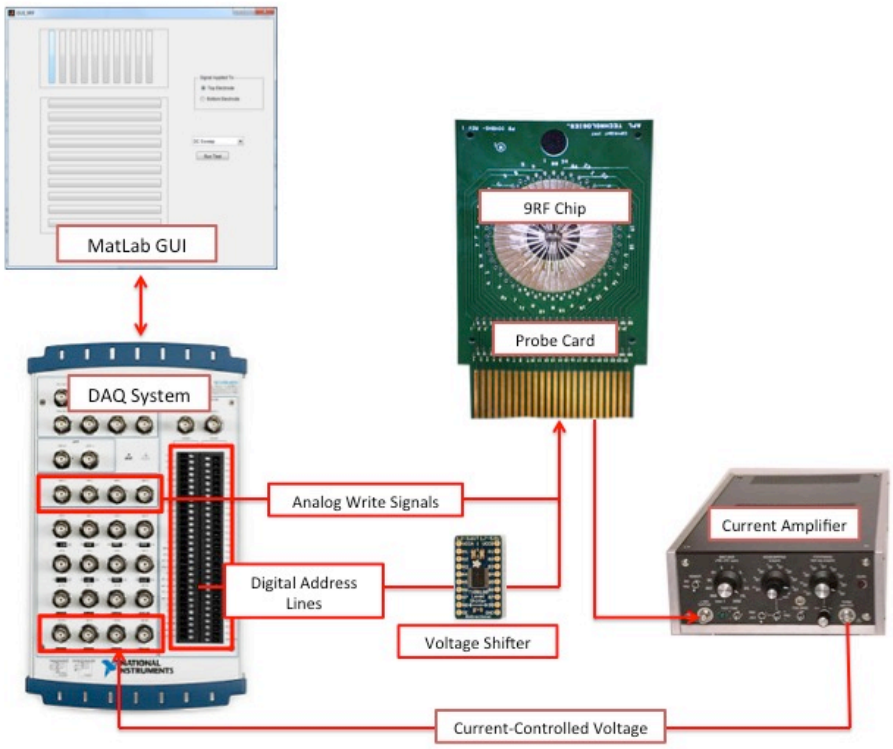


Figure 5.2: Test and Measurement setup for 7RF and 9RF memristor chips.

5.2 Array Testing Board Design

A special purpose board was designed to test memristor arrays in neuromorphic computing applications including the sparse coding tasks. The board is capable of applying timed voltage pulses and performing current measurements, with an integrated controller system to perform these tasks in an automated manner. Both the board and

controller designs have undergone several rounds of revisions as the project requirements were refined and deficiencies addressed.

A functional schematic of the board is given in Figure 5.3. The board was developed in collaboration with Professor Zhengya Zhang's research group at the University of Michigan. It can measure arrays in size of up to 32 rows and 32 columns. There are four digital to analog converters (DACs) capable of producing 0-5V independently. Two voltages are connected, through the matrix switches, to the rows, and two to the columns. The matrix switches are connected in such a way as to perform 2x32 routing, with a 32-bit binary word used to configure which of the rows (columns) is connected to DAC0 (DAC2) while the remaining rows (columns) are connected to DAC1 (DAC3). The board is capable of performing standard tests to characterize memristive devices including DC Sweeps, pulse measurements, and importantly, read and write procedures for memristor crossbar arrays.

A virtual ground with negative feedback is used to convert the current flowing to ground to a voltage that can be read by the analog to digital converters (ADCs). A variable resistor in the feedback back is used to control the amplification of the current signal. A multiplexer is included in the signal path to allow connection of either the virtual ground or the DAC. All control and data signals are passed through logic level converted to pin headers so the signals can be used off-board.

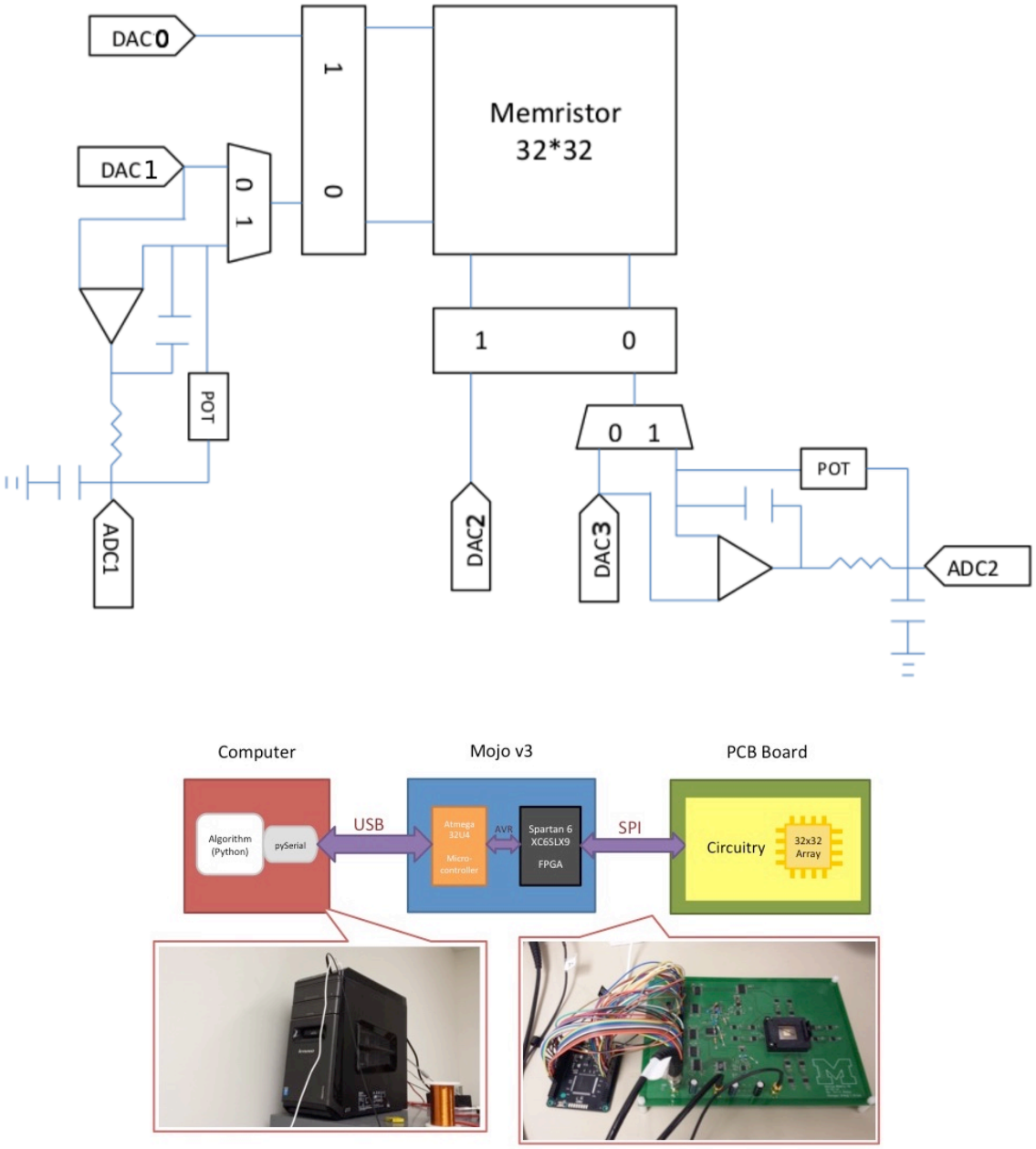


Figure 5.3: Functional schematic of the board (top) and complete system (bottom) of the test and measurement setup designed in collaboration the Zhang group.

5.3 Controller

The system controller was initially implemented as a finite state machine with a custom instruction set, detailed below. After completion, however, deficiencies were identified and the controller was redesigned using a soft microcontroller with custom peripherals. The main tasks of the controller were identified as:

- Load instructions
- Set DAC voltages
- Configure matrices
- Read ADC
- Store data
- Delay
- Transmit stored data

5.4 Finite State Machine

Initial designs were based on a finite state machine implemented on a Xilinx Spartan-6 field programmable gate array (FPGA). While basic operation is largely a sequential process, the particular design of the board demanded some parallel signals. For each task listed above, a module was implemented in Verilog that would perform the sequence of steps necessary to accomplish the task. A central control module was used to coordinate these actions using handshaking signals to determine when to proceed to the next task.

A primary benefit of this design is the speed of execution. A series of instructions are generated on the attached computer and downloaded to the controller. After receiving a start signal, the state machine will execute each instruction in order. Because the clock frequency is known and there are no branching or flow control instructions, execution time can be tightly controlled. This is particularly useful for creating sub-10ns pulse widths on the board. This tight timing control comes at the expense of code size as well

as difficulties in programming. Requiring strictly sequential instruction execution without the ability to branch precludes the use of looping structures or function calls. Thus, if some task needs to be repeated, the instructions are effectively inlined and require storage space for each repetition. Furthermore, because of the speed and synchronous nature of the block RAMs (BRAM) on the FPGA, the control unit was designed to have a 2 stage pipeline with instruction fetch and execution occurring in parallel. Unfortunately, not all instructions execute in the same number of clock cycles and so complicated handshaking is necessary to stall the pipeline for longer instructions. This proved to add unnecessary complications to the design and motivated the redevelopment of the control unit around a microprocessor-peripheral paradigm.

5.5 Microcontroller & Peripheral

5.5.1 Version 1

To address the challenges of flexibility and code size of the previous controller, a new controller was developed that uses a system-on-chip for modular design. Operations are initiated by a standard microcontroller, communicated with added peripherals via a shared bus. The use of a microcontroller allows flexibility for sequential programs, while the peripherals achieve the necessary parallelism for system control. A schematic of the control unit showing the functional units is given in Figure 5.4.

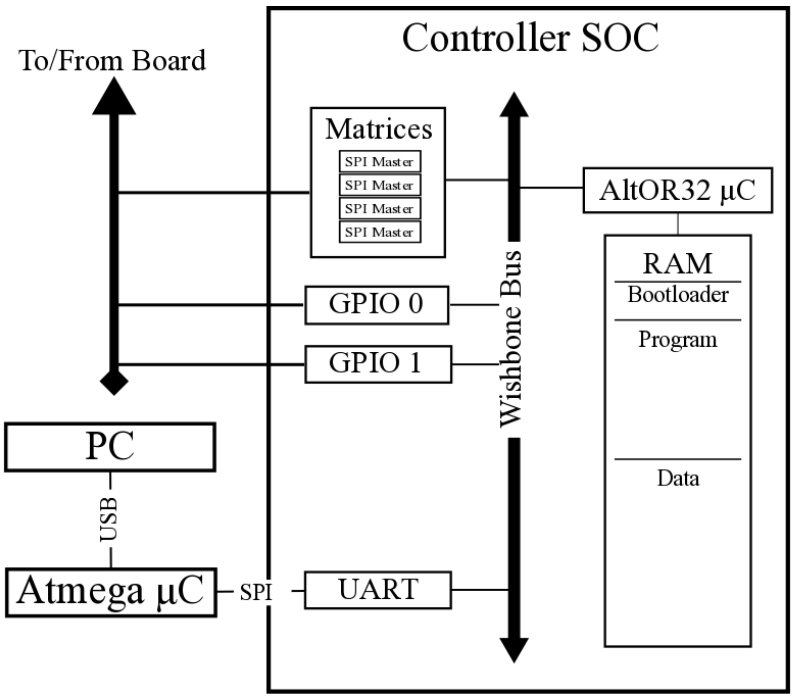


Figure 5.4: Test and measurement control system-on-chip targeted at a Spartan6 FPGA.

A soft microprocessor for implementation on the Xilinx FPGA was sought and the OpenRISC 1000 (OR1K) was investigated for use. After considering the limited resources of the Spartan6, a modified and stripped down version of the OR1K, known as the Alternative OpenRISC 1000, or AltOR32, was selected. This reduced version removes several instructions from the processor including the floating point unit, hardware multiplier, and pipeline delay slot. The AltOR32 used fewer than half of the available logic resources on the Spartan6 FPGA, leaving room for peripheral development as well as BRAM storage for collected data.

A general purpose input-output (GPIO) module was developed which could interface with the AltOR32 via the Wishbone bus. This enabled many of the test board functions to be implemented in software by simple reading and toggling appropriate control lines. The matrix control module from the previous control unit was recycled by

adding a Wishbone interface so that the AltOR32 could control the matrices via memory mapped registers. The handshaking required for this task was greatly simplified by having the processor repeatedly poll the module until its task completes.

The AltOR32 microcontroller executes standard RISC instructions, and the Gnu compiler toolchain (gcc) has been ported to the implementation. This allowed control functions to be implemented in standard C code and debugged via a UART peripheral. A five-stage pipeline, combined with compiler optimizations and garbage collection of unused functions during linking, helps keep code size to a minimum, while instructions execute nearly as fast as with the FSM design.

5.5.2 Version 2

Several shortcomings of the Version 1 test and measurement setup were identified and are briefly summarized below:

- DAC and op-amp offsets result in measured non-zero current even when input voltages are zero
- ADC resolution not sufficient to differentiate states when on/off ratio is reduced
- Communication delays too long when devices have limited retention
- Inability to apply negative voltage pulses

To address these issues, the test and measurement board was redesigned with the following changes:

- ADC resolution increased from 12 to 14 bits
- DAC resolution increased from 8 to 12 bits
- Additional multiplexor added to allow direct input of ground (rather than DAC output of 0V)
- Low-offset, unity-gain-stable op-amps used
- Dual supply DAC and matrix chips to allow negative voltage inputs
- High-speed USB2.0 communication interface for control board
- Onboard DRAM chip for pattern/data storage

A schematic of the new controller SoC is shown in Figure 5.5. To address the communication latency and throughput, the design was migrated to the OpalKelly XEM6010-LX45 board which offers a larger Spartan6 FPGA, significantly higher bandwidth via the Cypress USB to parallel conversion chip, and an integrated DRAM chip.

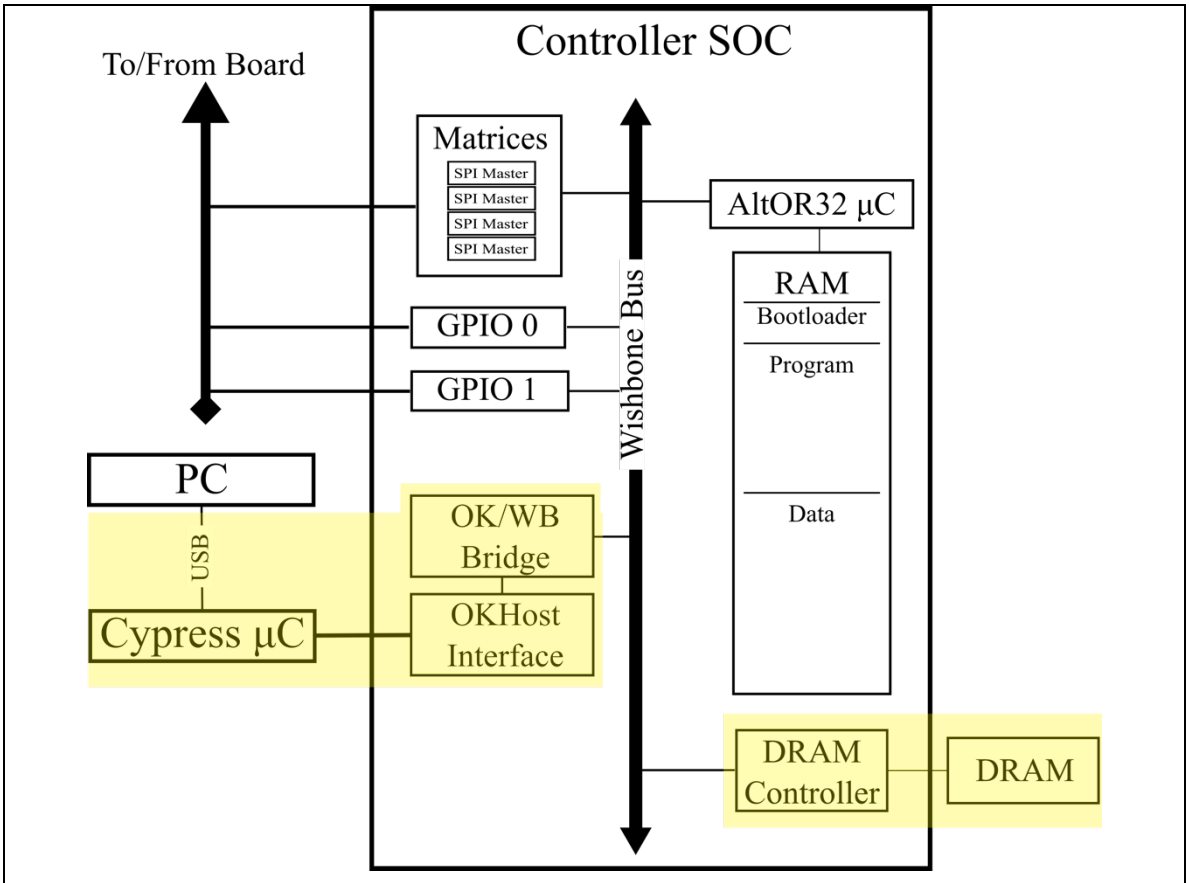


Figure 5.5: Schematic of new controller. The changes are highlighted in yellow.

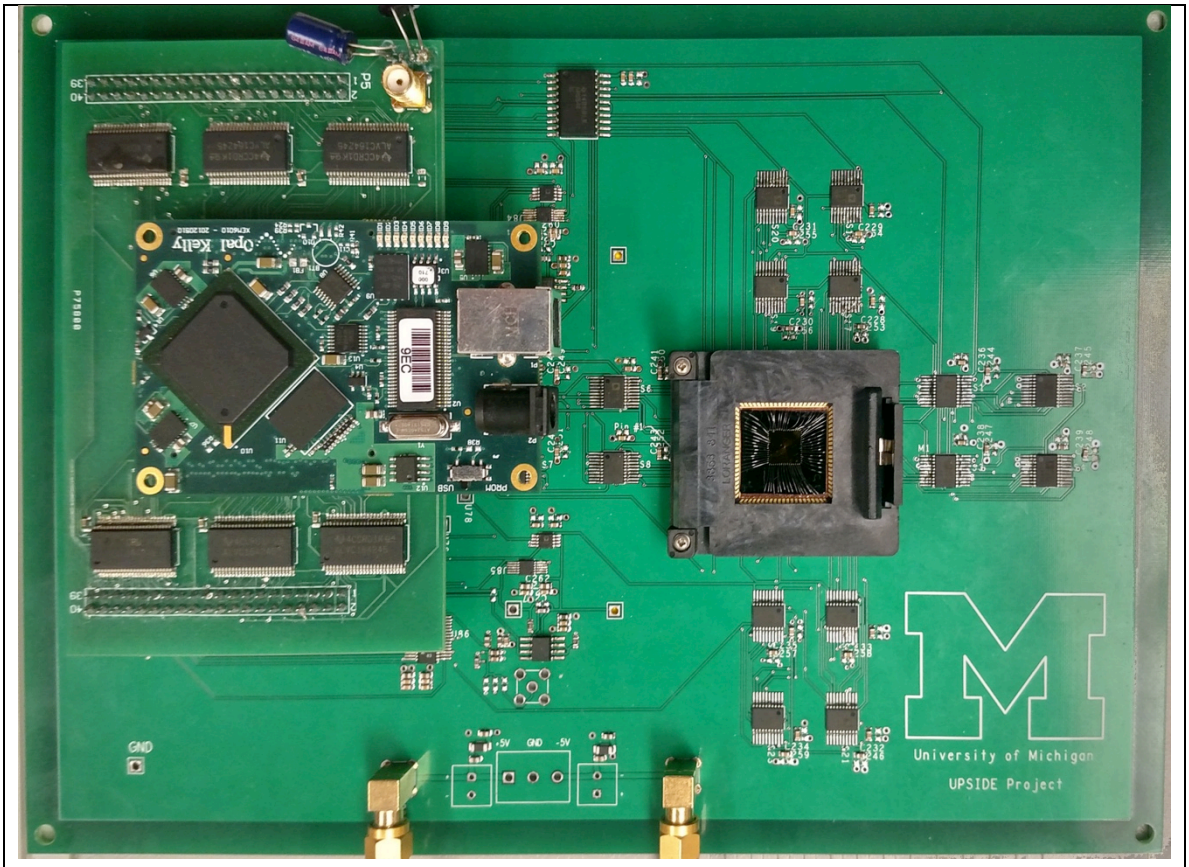
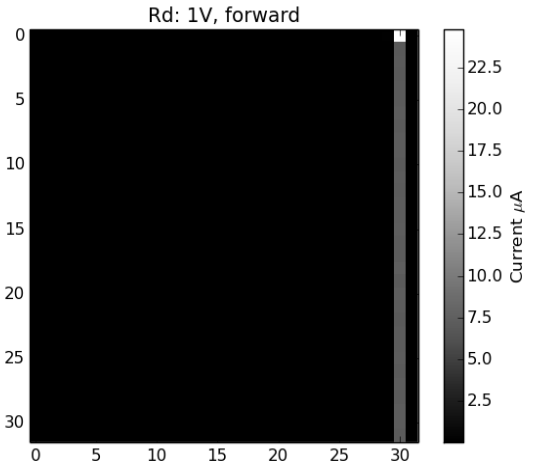


Figure 5.6: Test and measurement board revision. The board uses a new controller implemented with an Opal Kelly (left), and includes several upgraded components and design changes to address previous shortcomings.

The upgraded board design allows faster algorithm execution and higher training throughput by using larger on-board memories and a faster clock rate (up to 100MHz). Additionally the inclusion of multiplexers to allow the application of ground, as opposed to 0V DAC output, significantly reduces the problems associated with DAC and op-amp offsets as shown in Figure 5.7 below.



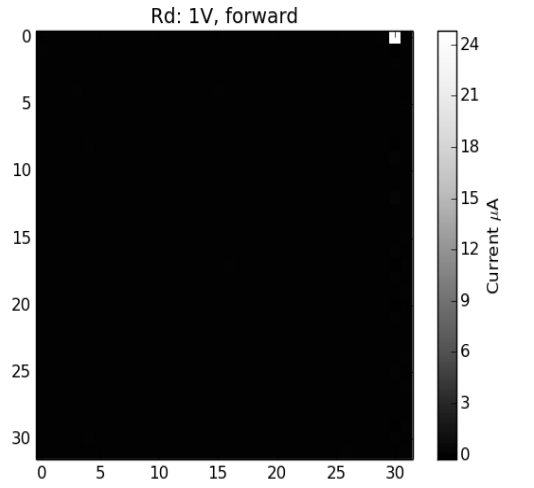


Figure 5.7: Effect of DAC offsets. A 1Kohm resistor is placed in the upper right corner of the array. While measuring each cell, the remaining rows are connected to a 0V DAC output (top) or directly to ground via a multiplexing input (bottom). Offset currents are eliminated by using an actual ground. Note the maximum measurable current is 24.8uA.

The top of Figure 5.7 demonstrates the problem of DAC/op-amp offsets when a short exists in the array. In this setup a 1KOhm resistor (a very low conductance compared to typical device resistances; typical of a shorted device) is placed in the upper right of the array and all other connections are completely open. Nonetheless, an appreciable current is apparent when measuring locations in the same column as the resistor short. This can

be understood by considering an offset in the DAC output. When measuring the second through thirty-second rows, an output of nominally 0V is applied to the first row (because it is not being read). However, a DAC offset of even a few millivolts will cause a current of several microamps to flow through the short, thus distorting the measurement for all locations that share the column. By passing an actual ground, rather than using the DAC, the offset is eliminated which greatly reduces the erroneous current.

The use of a popular and well-tested open source soft processor eliminated many of the shortcomings of the previous control unit design. With branching abilities, code size was greatly reduced which left more room for data storage. As a result, multiple readings could be made from the ADC to allow averaging, effectively reducing measurement noise.

5.6 Software Stack

The project code is written in a mixture of Python and C code. The Python functions direct the pre-processing and compilation of C routines and download the compiled binaries to the board. The generated data is received from within Python, averaged, and displayed with the Python-based Matplotlib library. Algorithm execution is directed by the Python code to reduce the processing load on the soft microcontroller, while board control routes benefit from the real-time execution of the microcontroller.

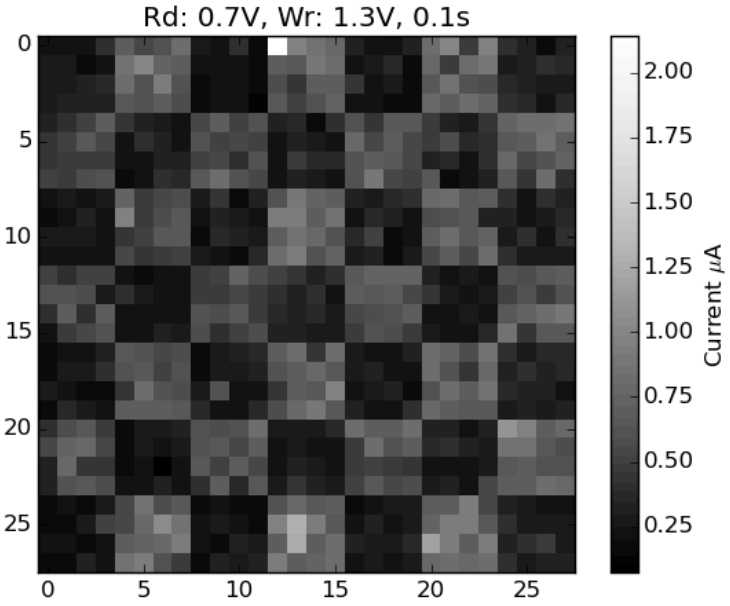
Low level board tasks such as setting the output voltages and configuring the matrix switches were written exclusively in C using memory-mapped control registers while higher level functions such as reading an array or programming a pattern were written in a mixture of C and Python. C code templates were developed to execute generic tasks. The Python code acted as a preprocessor for these templates, filling in

parameters such as hexadecimal values corresponding to a voltage or 32 bit configurations for the matrices. The SCons build tools are used to control compilation and linking which is performed by the or1knd-toolchain developed for the AltOR32.

5.7 Test and Measurement Results

Using the board test setup discussed above, a number of array measurements can be successfully performed. A few samples are included below to demonstrate the capabilities of the board. The current results show that, when using the memristor arrays as a memory, patterns can be effectively stored and retrieved. This servers a first step to demonstrating a learning system.

(a)



(b)

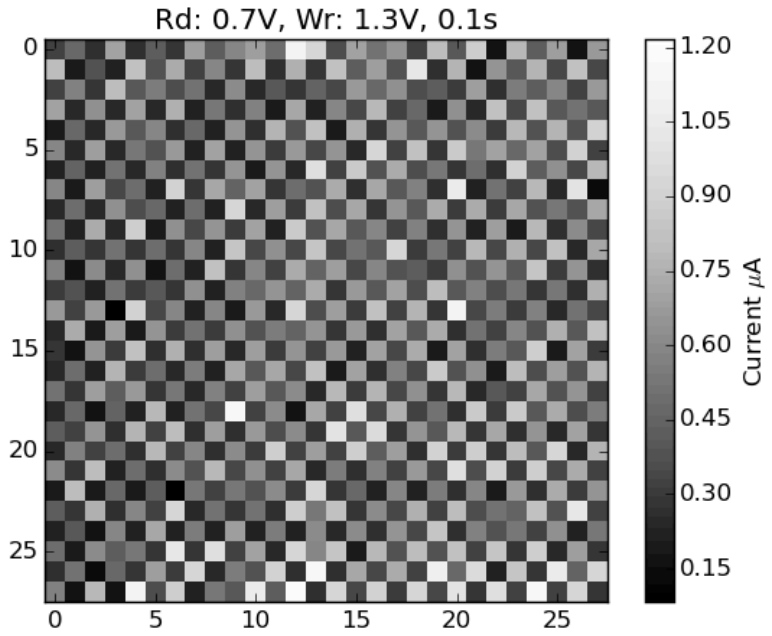


Figure 5.8: Binary checkerboard patterns stored in the array.

Figure 5.8 demonstrates that binary patterns (array element is either written or not) can be stored in the array and that there is a reasonable amount of uniformity between devices. Fig. 5.8(b) shows that small granularity can be achieved and that sneak-path currents are not significantly affecting the ability to distinguish off-cells from nearby on-cells.

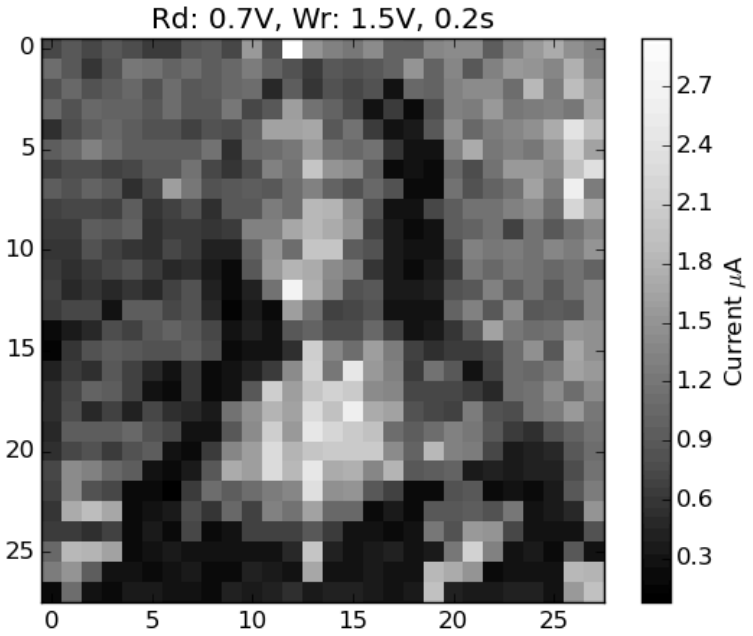


Figure 5.9: Mona Lisa pattern stored in the array.

Figure 5.9 demonstrates that, by varying the width of the programming pulse, the memristor cells can be effectively tuned to different conductance values to store a grey-scale image. No feedback mechanism or adaptive write pulse is necessary to achieve this effect.

5.8 Conclusions

Systematic simulations were performed to test the feasibility to achieve sparse coding and learning in memristor crossbar arrays. We should note that the memristor network can tolerate device variability in excess of 100%. However, the non-linear weight update can significantly deteriorate the network performance. By using a scheme to compensate the nonlinearity effects, successful learning and sparse coding can be achieved in simulations based on realistic device models even in the presence of realistic, large device variations. Several versions of test and measurement platforms were

developed and discussed. These measurement platforms allow for the experimental implementation of network-scale learning algorithms (discussed in the next chapters) in memristor crossbar arrays.

Chapter 6. Experimental Demonstration of LCA

There have been few experimental demonstrations of hybrid RRAM-computing at a network scale [84]–[87], and thus far, they have tended to be limited in size or functionality. In this chapter we describe the use of a 32x32 (1 kilobit) array to accelerate the computations of the locally competitive algorithm (LCA) as discussed in Chapter 4. A typical device fabrication flow is described followed by experimental results obtained using the array. LCA sparse coding is demonstrated and the effects of the dictionary size and choice of threshold parameter are investigated. The successful demonstration of memristor-based in-memory-computing at the network scale is an important milestone in the development of neuromorphic computing.

6.1 Device Fabrication

An important advantage of memristor technology is its compatibility with traditional CMOS devices. The low temperature fabrication process allows memristor devices to be integrated with fabricated circuits in a back-end-of-line process. This allows transistor-based technology to provide control signals for reading and writing operations for memory or neural circuit implementations. Additionally memristor devices can be stacked for increased density [35], [88], [89]. WO_x-based memristor arrays are used in this study. The fabrication processes include:

1. Starting with a non-conductive substrate. Typically Si wafer with 100-200nm grown SiO_2
2. W sputter deposition. Commonly 40 – 80 nm.

3. Patterning bottom electrodes. Typically done with e-beam lithography.
4. Ni hardmask deposition via e-beam evaporation. Thickness depends on W thickness, ~40nm for 60nm W.
5. Liftoff in Acetone or MicroChem Remover PG.
6. W etch using reactive ion etching. Etch chemistry is Cl_2 and O_2 gas mixture.
7. Ni hardmask removal in 1:1 HCl:DI solution for 30 minutes.
8. W oxidation to form WO_x via rapid thermal annealing. Temperatures range from 350 – 450 °C for 1 – 4 minutes depending on desired characteristics.
9. Patterning top electrodes. Typically with e-beam lithography.
10. Pd+Au top electrode deposition. Thickness must cover W/WO_x step height.
11. Liftoff in Acetone or MicroChem Remover PG.
12. WOx etch using top electrode as mask. Reactive ion etch with SF_6/C_4F_8 mixture.

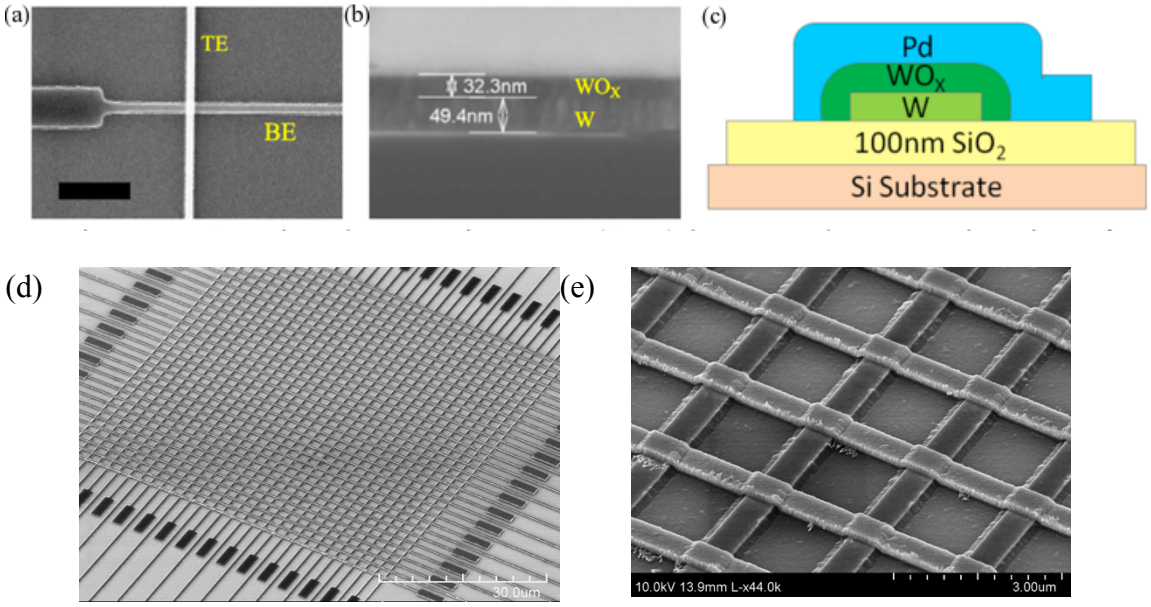


Figure 6.1: (a) SEM image of a single-cell device. (b) Cross section of the WO_x film. (c) Schematic cross-section of a complete device. (a-c). SEM images of completed crossbar arrays (d-e).

The basic device fabrication flow is relatively simple with the major steps consisting of the formation of a bottom electrode, an oxidation step to create the resistive

switching layer, and the formation of a top contacting electrode. The creation of the resistive switching oxide is a critical step in this process. The layer is formed by a timed rapid thermal annealing between 350°C and 450°C. The time and temperature determine the oxide thickness and material properties and have a strong influence on the resultant device behavior [83].

After fabrication, the memristor chip is placed in a chip carrier and the crossbar array's column and row electrodes are wire-bonded to the chip carrier's lead pads. Device characterization can then be conducted with the test and measurement board. Important parameters such as write, erase and read voltages, as well as write and erase durations must be determined in order to use the network for higher functioning. These parameters are then fed back into the simulation framework to update device models and network parameters to more accurately perform simulations.

6.2 Sparse Coding

6.2.1 Introduction

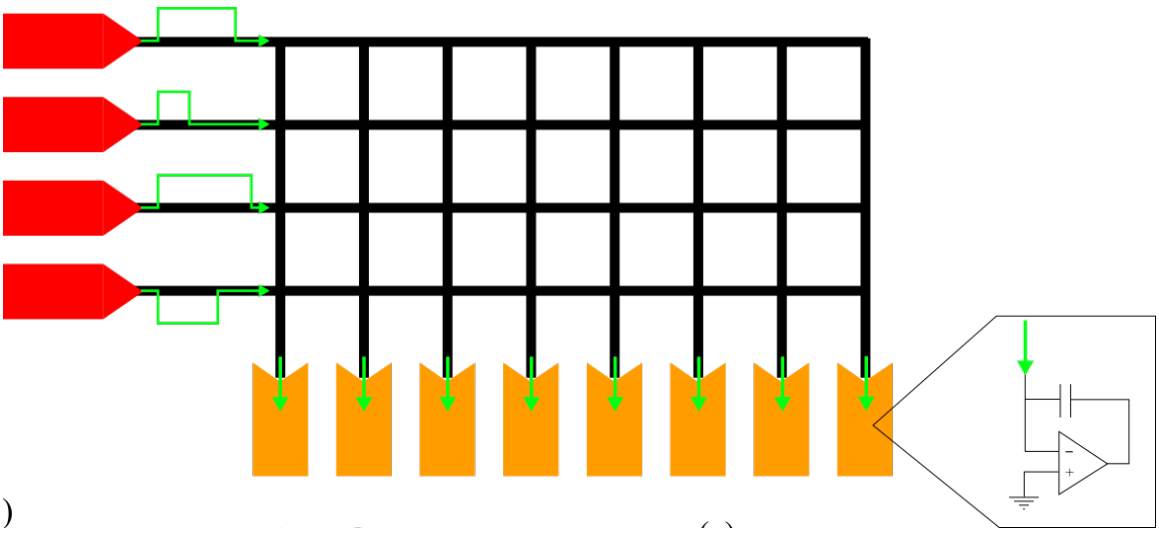
To demonstrate the full potential of the approach, the locally competitive algorithm (LCA) proposed in [61], [72] and discussed in Chapter 4, was implemented in a 32x32 WO_x-based memristor crossbar array. The algorithm is executed in a combined digital-analog computing approach. The memristor crossbar is used to accelerate the matrix-vector multiplications and matrix transpose operations in the analog domain, while neuron and 'residual' updates are conducted in the digital domain. Both the neuron update and system control functions are coordinated by a microcontroller and system-on-chip implemented on an FPGA as described in the previous chapter.

While a full description of the process is given in Chapter 4, Section 4, an outline of the procedure is given below for reference. The network is organized such that the residual is input on the rows of the matrix, while the neurons are positioned on the columns, as shown in Fig. 6.2. The governing equation for neuron dynamics, presented as (4.3) is repeated below for reference:

$$\frac{du}{dt} = \frac{1}{\tau}(-u + (p - \Phi \cdot a^T)^T \cdot \Phi + a) \tag{6.1a}$$

$$= \frac{1}{\tau}(-u + (p - \hat{p})^T \cdot \Phi + a) \tag{6.1b}$$

$$= \frac{1}{\tau}(-u + r^T \cdot \Phi + a) \tag{6.1c}$$



)

Figure 6.2: LCA network schematic. Residual drivers on the rows indicated in red. Neurons on the columns indicated in orange. Inset shows a leaky integrative neuron.

The LCA is implemented as a sequence of forward-input pulses to calculate the input to the neurons followed by a sequence of backward-input pulses to calculate the estimated (or reconstructed) input. The forward-input operation effectively calculates the matrix-vector product of the network weights with the input (residual), $r^T\Phi = (p - \hat{p})\Phi$.

Similarly the backward-input pulses calculate the matrix-vector product of the transpose of the network weights with the neuron activities, Φa^T. The backward-input operation is accomplished by reversing the positions of the drivers and integrators between the rows and columns, respectively.

The network dynamics are such that neurons with receptive fields (the column of weights connected to it) that better match a feature in the input will be charged faster. Once these neurons' membrane potentials reaches above a threshold (set by λ), they become active and begin to contribute to the reconstruction of the input, $\hat{p}$. Because the network is driven by the difference between the original input and the reconstruction, $r = (p - \hat{p})$, when a neuron becomes active, it effectively removes the feature from the residual—the driving inputs of the network. By doing so, the neuron deprives other neurons of the input current associated with that feature, effectively suppressing their activation. In this manner, the network implements competition through the inhibition term in Eqn. 6.1(a), which prevents over-representation of the features and enables better reconstruction of the input.

6.2.2 Critically Complete and Over Complete Dictionaries

The dynamics of this process can be seen in Fig. 6.3. In this experiment, a dictionary, shown in Fig 6.3(a), is programmed into the columns of the array. Each 5x5 pixel dictionary element corresponds to a single column of the array. Hence the crossbar array required to hold this dictionary will have dimension 25x10. Next, an input is constructed using a linear combination of a subset of the dictionary elements; this experiment used dictionary element 0 + element 4 and the resultant input is shown in Fig 6.3(b). Plotted in Fig. 6.3(d) is the membrane potentials of all neurons after each iteration

of the forward-backward LCA algorithm. The threshold for neuron activation is plotted as a horizontal line and is set to 60.

From the figure it can be seen that for the first few cycles, no neuron is above threshold and so all the neurons are steadily charging. At this point it is important to note that even neurons with receptive fields that share no pixels in common with the input (neurons 1, 2, and 3) are also being charged (although at reduced rates). While this would not occur in an ideal setup, in the experiment some non-zero currents will still pass through devices even in the minimal conductance state (since the minimal conductance of the memristors are ~0.5uS and are non-negligible), causing the charging of the neurons, albeit to a lesser extent than those neurons that do have pixels in common with the input.

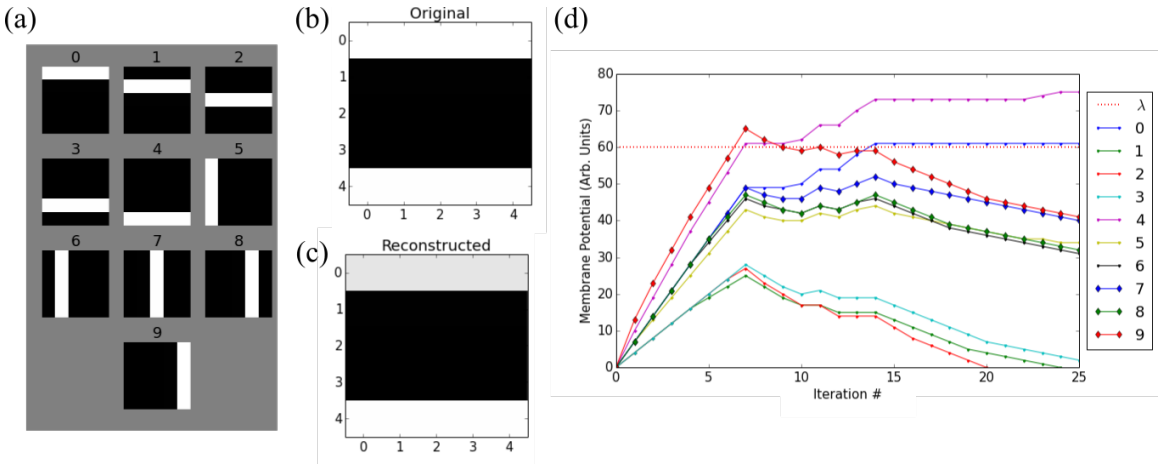


Figure 6.3: Sparse coding with a nearly critically complete dictionary. The algorithm activates the two neurons that represent the input.

After approximately the 7th iteration, two neurons, neuron 9 and neuron 0, cross the threshold and begin to contribute to the reconstruction. When neuron 0 becomes active, it effectively removes the top horizontal bar from the residual input, depriving all other neurons of this input. This has an immediate effect on neuron 9: deprived of it's

input (likely the upper right corner pixel), the membrane potential quickly begins to decay and once it crosses below the threshold, it no longer contributes to the reconstruction. At this point, however, the bottom horizontal bar is still not being represented in the reconstruction, or in other words, its pixels remain in the residual, continuing to charge neurons with overlapping receptive fields. At approximately iteration 14, neuron 4's membrane potential has charged sufficiently for it to begin contributing to the output. When it does so, the residual is greatly reduced and the neurons no longer continue to charge. As can be seen on the right side of Fig. 6.3(d), the active neurons have reached a comparative equilibrium and thus the coding task is now complete. The reconstructed input can then be created by summing the receptive fields of the two active neurons (neurons 0 and 4), weighted by their activities, as shown in Fig. 6.3(c). The reconstruction correctly reproduced the original input and at the same time identified the two major features from the dictionary (neurons 0 and 4). Ideally, the activity of neurons 0 and 4 should be identical, since the input was simply the sum of two horizontal bars. As can be seen that Fig 6.3(c), differences in the experimentally observed neuron activities (Fig 6.3b) lead to a reconstruction that is not a perfect match of the original input. This can be attributed to device variations and possibly also may have resulted from the discretized nature of the input (due to the limited resolution of the microcontroller when calculating the charges).

This first experiment used a limited dictionary size of 10 neurons and their associated receptive fields. While some neurons share an overlap of pixels from other neurons, the amount of overlap is minimal. In fact, while there are 25 input pixels (thus 25 input rows), the dimensionality of the input vector space is reduced to 9 by limiting

inputs exclusively to horizontal and vertical bars and their linear combinations (the dimension can be determined by the matrix rank of dictionary). Thus, with an input dimensionality of 9, and an output dimensionality 10, the dictionary can be said to be only marginally *over-complete* (if the input and output dimensions are equal, the dictionary is said to be *critically complete*). The solution to sparse coding with a critically complete dictionary is significantly easier than with an over-complete; an optimal solution can be found analytically and forms the basis of *principal component analysis*. However, a more sparse representation can be obtained by using over-complete dictionaries. Over-complete dictionaries allow neurons to 'specialize' their receptive fields to a particular feature rather than seeking an average of potentially several distinct features.

In the next experiment, we study sparse coding using an over-complete dictionary (dictionary size 25x20). The same input, reproduced as Fig. 6.3(c) is used to test the ability of the over-complete dictionary to choose a more efficient representation.

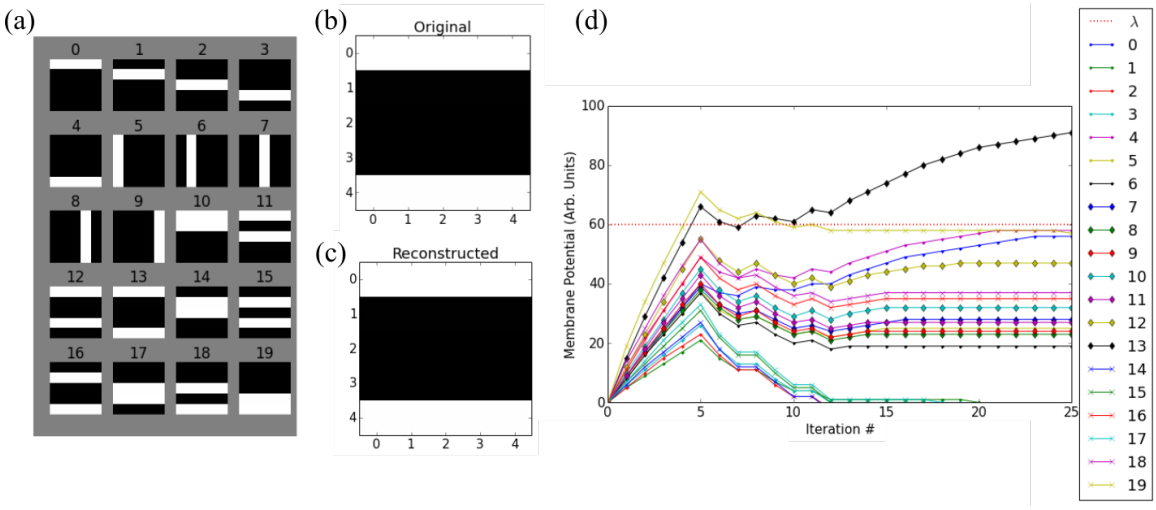
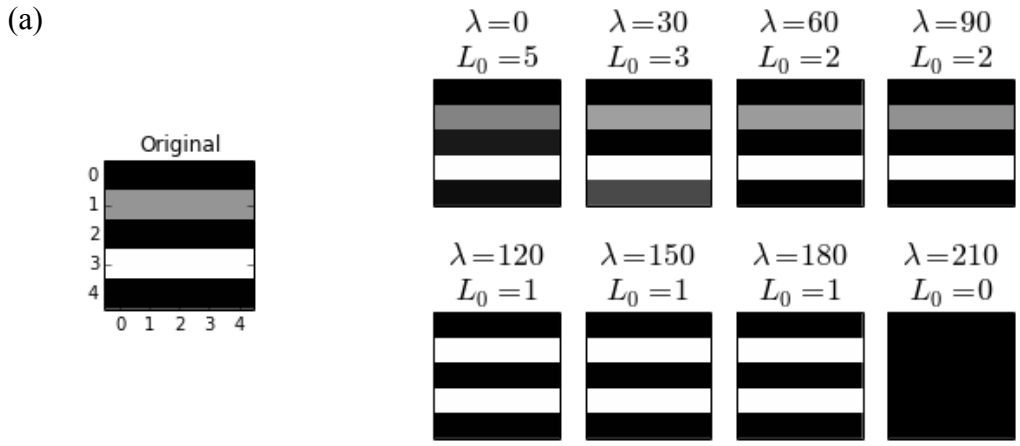


Figure 6.4: Sparse coding with an over complete dictionary. The algorithm activates only one neuron to represent the input more sparsely.

In this experiment, the dictionary has been expanded to include all $\binom{5}{2}$ doublet horizontal bars; the full dictionary is shown Fig. 6.4(a). With 20 total dictionary elements, the dictionary is now more than two times over complete (again, considering the input dimensionality to be 9). From the membrane potential plot in Fig. 6.4d, it can be seen that, like in the previous experiment, many of the neurons begin charging until one or more neurons crosses the threshold, λ. In this case neurons 13 and 19 become active and begin contributing to the reconstruction. At this point (iteration number 5), both neurons subtract the bottom horizontal bar from the input. In this case, the feature is being over-represented, and thus the residual for these pixels will be negative. A negative input residual causes matching neurons to discharge faster due to the $r^T\Phi$ term in the neuron dynamics Eqn. (4.3). This causes neuron 19 to sink below the threshold and become inactive; once this occurs, the feature is no longer over-represented and the membrane potential of neuron 19 settles to just below the threshold. Since neuron 13 in the new dictionary is able to completely capture the input pattern, it becomes the sole active neuron after the network has stabilized. The reconstructed image can now be created by the receptive fields weighted by the activities of the output neurons (in this case only neuron 13) and shown in Fig. 6.4(c). The ability for the network to identify a more sparse representation, despite the initial solution suggesting otherwise, is a key feature of sparse coding. In this case with an over-complete dictionary a more sparsely encoded solution can indeed be identified than in the case with the smaller dictionary (Fig. 6.3), after the network reaches a stable state and despite that initial solutions suggesting otherwise.

6.2.3 Effect of Threshold Parameter, λ

Since the neurons do not become active or contribute to the reconstruction until their membrane potentials have crossed the threshold parameter, λ, this parameter can be used to control the sparsity of the final solution, with a high λ preventing weakly matching or redundant neurons from becoming or remaining active. This effect was observed in Fig 6.4: the activation of neuron 13 more closely matched the input and thus forced neuron 19 below the threshold, thus increasing the sparsity of the solution. To examine more thoroughly the effect that λ has on neuron dynamics, an experiment was performed where the dictionary and input pattern was kept constant while the λ parameter was varied from 0 to 240. The results, including the reconstructed image and number of active neurons as a function of λ, are presented in detail in Fig. 6.5, while detailed neuron membrane potential dynamics at different λ values are shown in Fig. 6.6.



(b)

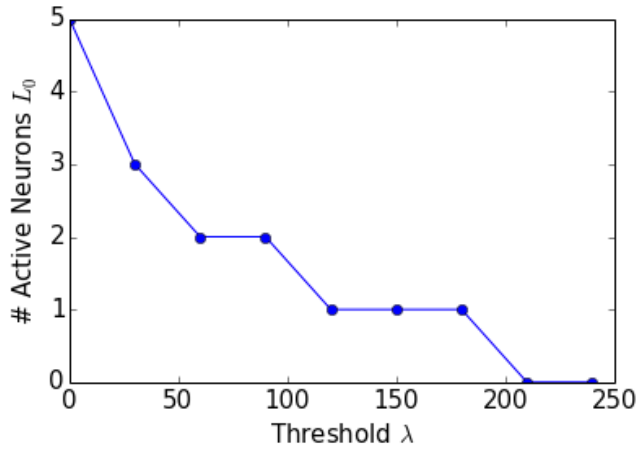


Figure 6.5: The impact of the threshold parameter on sparse coding. As λ is increased from 0 to 240 in increments of 30, the reconstructions in general become less faithful to the original input (a) and more sparse (b). Successful reconstruction, however, is not extremely sensitive to the choice of λ.

Several interesting conclusions can be drawn about the impact of the threshold parameter on the neuron dynamics. Perhaps most intuitive result is that increasing λ results in fewer neurons being activated as shown in Fig. 6.5. With the leak and competition terms pushing the membrane potential down, a higher threshold makes it harder for neurons to activate. Interestingly, however, the network dynamics are not extremely sensitive to λ (as long as λ is within a certain range) and increasing the threshold above a neuron's existing steady state value does not automatically cause the neuron to become inactive. For example, we can see that increasing the threshold from 120 to 180 did not reduce the sparsity of the solution or change the reconstruction. This is despite the fact that, in the case of $\lambda = 120$, the final values of the active neurons is only ~157 (Fig. 6.6). Put differently, one might naively expect that no neurons would be active when the threshold is increased to $\lambda = 180$; in fact, this is not the case. Instead, the network adapts to the higher threshold by allowing neurons to be charged and settle at a higher value. When the

threshold is raised further, however, different dynamics may emerge since different neurons charge at different rates depending on how well their receptive fields match the input pattern, and a new sparse solution may be obtained, as shown in Figs 6.5 and 6.6 where the previously active neurons no longer become active and new solutions (neuron 15) are found.

This behavior can be further explained by examining Eqn. (4.1), repeated below for reference:

$$\min_{a,\Phi}(\|\Phi a^T - p\|_2^2 + \lambda\|a\|_0) \tag{6.2}$$

From [61], we know that the algorithm is attempting to minimize Eqn. (6.2), with the second term in the cost function corresponding to a sparseness penalty. The exact value of λ determines the relative importance of accurately reconstructing the input versus achieving a sparse representation. Increasing λ increases the penalty for activating each additional neuron, but it does not necessarily tip the balance of Eqn. (6.2) in favor a suppressing a neuron. Instead, an increase in λ may simply cause the neurons' membrane potentials to rise (causing a corresponding rise in their activations) in order to remain active. This has the effect of over-representing features in the input, but the relative penalty for it can be less than suppressing a neuron from being active altogether. Further increases in λ can result in too high a sparsity penalty to keep a neuron active and so the network will favor a new, more sparse representations as seen when λ increases from 90 to 120 and again when it increases from 180 to 210 (in the latter case, the sparseness penalty is so high, it becomes more favorable to activate no neurons—a blank sparse representation).

The non-linear behavior of the network dynamics makes it hard to predict what value of λ to choose to achieve a given sparsity. Fortunately, as can be seen in Figs. 6.5 and 6.6, the network dynamic is not extremely sensitive to the exact value of λ as long as λ is within a proper range. Additionally, there is a clear relationship between the magnitude of the input and λ. For example, if we have achieved an acceptable sparse coding of input p using threshold λ and then decide to scale the magnitude of p by 100 ($p' = 100p$), then we should scale λ by a similar amount ($\lambda' = 100\lambda$) to achieve the same sparsity. This effect is due to the fact that no normalization was performed for the input in our experiments. This property is evident from Eqn. (6.2): λ must be scaled to keep the sparseness penalty proportional to the reconstruction error term.

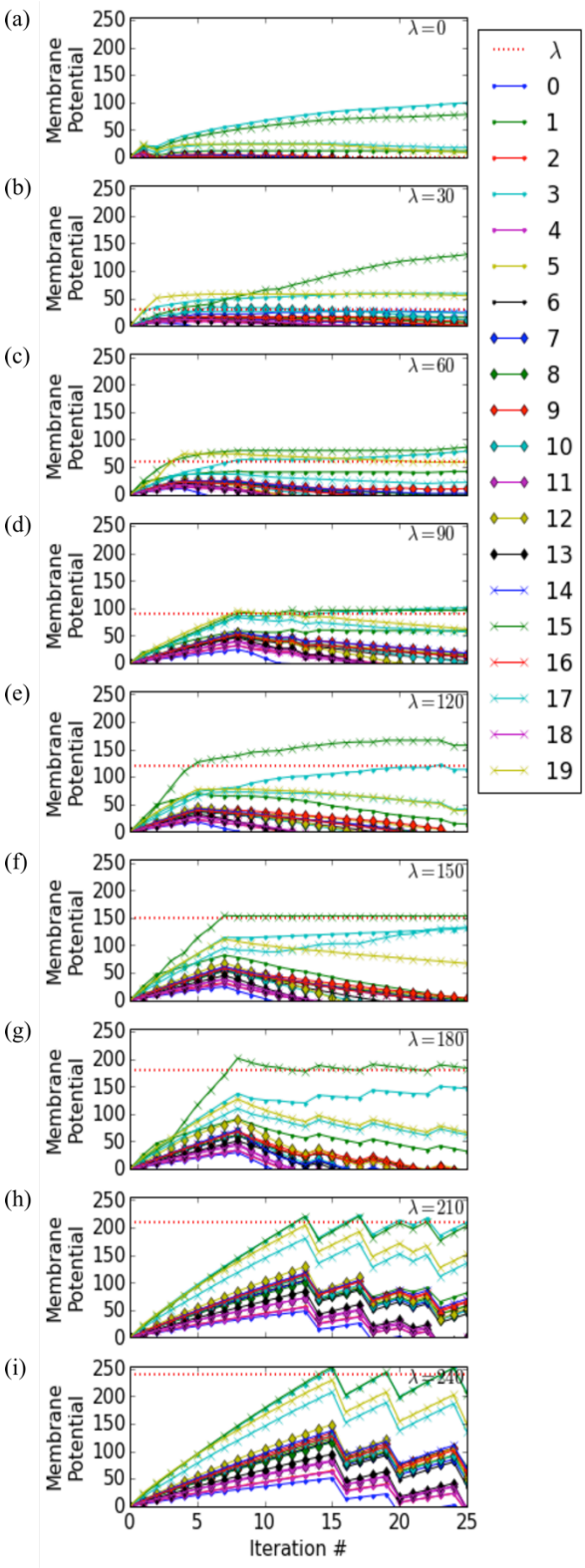


Figure 6.6: The impact of threshold on neuron dynamics. λ is increased from 0 to 240 in increments of 30 (a-i).

6.3 Conclusion

A WO_x-based memristor crossbar array was fabricated and tested to implement the sparse coding algorithm using LCA. It is important to note that all fabrication steps occur below 450°C which allows the process to be integrated in a back-end-line manner with CMOS technologies [90]. Furthermore, the introduction of new materials into a fab environment is a costly process due to the need for extensive process development and risk of contamination [91]. Tungsten, however, has been used extensively in standard CMOS chips and thus is a good candidate for integration [90]. As discussed in Chapter 5, Section 1, successful integration with 180nm and 90nm CMOS chips has already been demonstrated and future work will look at implementation of a larger array.

The locally competitive sparse coding algorithm, presented in Chapter 4, was experimentally demonstrated using the fabricated array with the test and measurement platform presented in Chapter 5. Programmable device behavior was confirmed, allowing a pre-computed dictionary to be stored in the array. The success of the algorithm execution serves as a proof-of-concept for the matrix-vector multiplication and in-place matrix-transpose operations. The robustness of the algorithm to tolerate device non-idealities was confirmed as well as the predicted response to an increasing sparseness penalty. It is interesting to note that many of the neuron membrane potentials tend to settle very close to the threshold; this is indicative of the driving forces in the sparse coding algorithm and suggests that a careful or contrived choice of threshold parameter is not necessary. The results are the first known example of sparse coding with resistive switching devices and one of the largest scale examples of neuromorphic computing with memristors to date.

Chapter 7. Conclusions

7.1 Summary

In this work we developed the theoretical and measurement frameworks for neuromorphic computing applications using resistive switching devices and demonstrated functions such as learning vector quantization and sparse coding in memristor crossbar arrays through simulation using realistic devices models and experiments using nanofabricated memristor crossbar arrays.

Starting from device developments, we fabricated resistive switches with analog switching characteristics that can be used in a crossbar network. Physics-based and compact models were developed to capture both the static conductance as well as the dynamic switching behaviors of the device. The models provided an understanding of the physical phenomena underlying the resistive switching effects as well as a means to perform realistic, yet tractable network simulations. Subsequently, our network-level simulations based on the compact model demonstrated the usefulness of resistive switching crossbar arrays in accelerating matrix-vector multiplication and learning operations. The development of a modular, multithreaded simulation framework allowed for several algorithms, including vector quantization and the locally competitive algorithm (LCA), to be combined with learning mechanisms to test the feasibility of experimental hardware implementations.

Finally, a test and measurement system was constructed to enable implementation of a sparse coding algorithm on actual resistive switching hardware. The algorithm was

modified to run on fixed point hardware using crossbar arrays of resistive switches for weight accumulation, storage and matrix-vector product acceleration. Using the system, the sparse encoding of simple bar patterns was successfully demonstrated. These experiments are the first known demonstration of a neuromorphic sparse coding platform using resistive memristive elements. The results show that resistive switches can not only be used for data storage, but can allow combined memory and computing operations that lead to significant acceleration in neuromorphic applications.

7.1.1 Device Modeling

The modeling work presented in this dissertation provides both an understanding of the physical processes underlying resistive switching phenomena as well practical tools to guide the design and simulation of crossbar arrays of these devices. Chapter 2 introduces a detailed dynamic memristor model based on a metal cation based system using Ag/α-Si. The model is able to accurately capture switching dynamics and explain the apparent voltage threshold for device switching. The work resulted in a journal publication [43] and a stochastic extension of the model was used for developing the experiments in another journal paper [47].

A similar model was developed to describe resistive switching in non-stoichiometric tungsten oxide devices based on the electric-field induced movement of oxygen vacancies. The work resulted in two journal publications [49], [92] and was used as a foundation for development of the learning networks presented in the remainder of the dissertation. In Chapter 4 the model was extended to consider device variability, noise, and device failures and was used in developing ways to address these nonidealities.

7.1.2 Neuromorphic Algorithms

Chapters three and four discuss the co-development of algorithms and architectures for performing neuromorphic computing using memristor crossbar arrays. Learning vector quantization (LVQ) is used as a case study in Chapter 3 to discuss how memristors can be used to accelerate accumulation and matrix-vector operations in a parallel manner. The Winner-take-all—Oja's learning rule is presented as an unsupervised, constrained-Hebbian rule amenable to implementation in crossbar arrays. Results demonstrating the use of memristive LVQ to classify handwritten digits from the MNIST database were presented and published in a conference proceeding [93].

Chapter four makes use of the crossbar acceleration techniques, developed in the previous chapter, to implement an adaptive sparse coding algorithm. The Locally Competitive Algorithm (LCA) by Rozell [72] was chosen for its advantages over competing approaches, its biological inspiration, and its adaptability to a memristive hardware implementation. A modular array simulation framework was developed to test and simulate sparse coding as well as dictionary learning approaches. Our system-level simulation results based on realistic device models demonstrate the feasibility of accelerating LCA with memristor crossbar arrays and the impacts of device non-idealities on algorithm performance. The work highlighted the importance of incorporating feedback mechanisms in neuromorphic computing to compensate device parameter variations. Specific techniques including adjusting write durations and serially connected anti-fuses were proposed to address issues related to device programming nonlinearity and conductive device failures. The simulation work has resulted in several journal and

conference publications [93], [94] and another journal paper just accepted for publication in *IEEE Transactions on Neural Networks and Learning Systems* [95].

7.1.3 Experimental Demonstration of Sparse Coding in Memristor Crossbars

A programmable test and measurement setup for characterizing single cell and crossbar arrays of resistive switches was designed and constructed and is described in Chapter 5. A combination of Verilog, C, and Python was used to build a system-on-chip controller that allowed for device programming and algorithm execution in a real-time manner. The LCA algorithm with crossbar acceleration discussed in Chapter 4 was migrated to a fixed-point implementation for execution with the system. By using an over-complete dictionary set, we show experimentally in Chapter 6 that the memristor network can effectively implement sparse coding by minimizing an energy function that includes both the error term and sparsity penalty. The neuron membrane potential dynamics were systematically analyzed as a function of input patterns and the threshold parameter. A journal paper based on studies in Chapter 6 is currently in preparation. These experimental results serve as a proof-of-concept demonstration for using memristor crossbar arrays to combine memory and computing elements and achieve efficient computation with high parallelism. The successful application of this approach to the task of signal sparse coding will also motivate future experimental for a more tightly integrated system and to reduce power and area while improving signal throughput.

7.2 Future Work

The next step in the research is to implement in-situ network learning using the winner-take-all strategy combined with Oja's rule as described in Chapter 4. A successful

implementation of this learning would constitute a complete, adaptable sparse coding system in memristor hardware. Further developments will focus on the supporting circuitry and network size scaling. As a first step, the test and measurement board will be condensed into a single CMOS chip. This will provide improved performance by reducing parasitic capacitances and reduce training and inference time. The area and energy usage can be further improved by converting many of the neuron circuits, currently implemented in digital CMOS, to analog circuitry. Larger memristor arrays, (e.g. 7.2 kilobit) will also be fabricated. The 49x147 array can be used to process 7x7 sized-patches while achieving 3x over-completeness. We believe such a network size can be used to effectively process natural images. Such demonstrations, combined with continued device optimizations to improve update linearity, analog switching dynamic range, device variability, yield and switching current, will form significant steps forward to bring efficient, high-performance memristor-based neuromorphic computing systems to reality.

References

[1] J. Von Neumann, “First Draft of a Report on the EDVAC,” *IEEE Ann. Hist. Comput.*, no. 4, pp. 27–75, 1993.

[2] J. Backus, “Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs,” *Commun. ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[3] H. Markram, “The human brain project,” *Sci. Am.*, vol. 306, no. 6, pp. 50–55, 2012.

[4] J. Von Neumann and R. Kurzweil, *The computer and the brain*. Yale University Press, 2012.

[5] R. Sarpeshkar, “Brain power-borrowing from biology makes for low power computing [bionic ear],” *Spectr. IEEE*, vol. 43, no. 5, pp. 24–29, 2006.

[6] G. Indiveri and T. K. Horiuchi, “Frontiers in neuromorphic engineering,” *Front. Neurosci.*, vol. 5, 2011.

[7] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, and others, “Neuromorphic silicon neuron circuits,” *Front. Neurosci.*, vol. 5, 2011.

[8] R. Douglas, M. Mahowald, and C. Mead, “Neuromorphic analogue VLSI,” *Annu. Rev. Neurosci.*, vol. 18, pp. 255–281, 1995.

[9] C. Mead, “Neuromorphic electronic systems,” *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[10] R. Sarpeshkar, “Analog versus digital: extrapolating from electronics to neurobiology,” *Neural Comput.*, vol. 10, no. 7, pp. 1601–1638, 1998.

[11] G. Indiveri and S.-C. Liu, “Memory and information processing in neuromorphic systems,” *ArXiv150603264 CsNE*.

[12] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, “Graphlab: A new framework for parallel machine learning,” *ArXiv Prepr. ArXiv14082041*, 2014.

[13] J. Gibbons and W. Beadle, “Switching properties of thin NiO films,” *Solid-State Electron.*, vol. 7, no. 11, pp. 785–790, 1964.

[14] D. Lamb and P. Rundle, “A non-filamentary switching action in thermally grown silicon dioxide films,” *Br. J. Appl. Phys.*, vol. 18, no. 1, p. 29, 1967.

[15] D. Ielmini, S. Spiga, F. Nardi, C. Cagli, A. Lamperti, E. Cianci, and M. Fanciulli, “Scaling analysis of submicrometer nickel-oxide-based resistive switching memory devices,” *J. Appl. Phys.*, vol. 109, no. 3, p. 034506, 2011.

[16] S. H. Jo and W. Lu, “Ag/a-Si: H/c-Si resistive switching nonvolatile memory devices,” in *Nanotechnology Materials and Devices Conference, 2006. NMDC 2006. IEEE*, 2006, vol. 1, pp. 116–117.

[17] S. H. Jo and W. Lu, “CMOS compatible nanoscale nonvolatile resistance switching memory,” *Nano Lett.*, vol. 8, no. 2, pp. 392–397, 2008.

[18] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[19] L. O. Chua, “Memristor-the missing circuit element,” *Circuit Theory IEEE Trans. On*, vol. 18, no. 5, pp. 507–519, 1971.

[20] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” *Solid-State Circuits IEEE J. Of*, vol. 41, no. 3, pp. 712–727, 2006.

[21] P. Beckett and A. Jennings, “Towards nanocomputer architecture,” *Aust. Comput. Sci. Commun.*, vol. 24, no. 3, pp. 141–150, 2002.

[22] “Samsung Starts Mass Producing Industry’s First 3D Vertical NAND Flash,” 06-Aug-2013. [Online]. Available: http://www.samsung.com/semiconductor/insights/news/12990.

[23] “Micron and Intel Unveil New 3D NAND Flash Memory,” 26-Mar-2015. [Online]. Available: http://newsroom.intel.com/community/intel_newsroom/blog/2015/03/26/micron-and-intel-unveil-new-3d-nand-flash-memory.

[24] Y. Park, J. Lee, S. S. Cho, G. Jin, and E. Jung, “Scaling and reliability of NAND flash devices,” in *Reliability Physics Symposium, 2014 IEEE International*, 2014, p. 2E–1.

[25] "Micron Announces Availability of Phase Change Memory for Mobile Devices," 18-Jul-2012. [Online]. Available: http://investors.micron.com/releasedetail.cfm?releaseid=692563.

[26] H.-S. P. Wong and S. Salahuddin, "Memory leads the way to better computing," *Nat. Nanotechnol.*, vol. 10, no. 3, pp. 191–194, 2015.

[27] L. O. Chua and S. M. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[28] L. Chua, "Resistance switching memories are memristors," *Appl. Phys. A*, vol. 102, no. 4, pp. 765–783, 2011.

[29] M. N. Kozicki, M. Park, and M. Mitkova, "Nanoscale memory elements based on solid-state electrolytes," *Nanotechnol. IEEE Trans. On*, vol. 4, no. 3, pp. 331–338, 2005.

[30] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nat. Mater.*, vol. 6, no. 11, pp. 833–840, 2007.

[31] S. H. Jo, K.-H. Kim, and W. Lu, "High-density crossbar arrays based on a Si memristive system," *Nano Lett.*, vol. 9, no. 2, pp. 870–874, 2009.

[32] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[33] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive'switches enable 'stateful'logic operations via material implication," *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.

[34] J. Borghetti, Z. Li, J. Straznicky, X. Li, D. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proc. Natl. Acad. Sci.*, vol. 106, no. 6, pp. 1699–1703, 2009.

[35] S. Gaba, P. Sheridan, C. Du, and W. Lu, "3-D Vertical Dual-Layer Oxide Memristive Devices," *Electron Devices IEEE Trans. On*, vol. 61, no. 7, pp. 2581–2583, 2014.

[36] M. Lee, C. Lee, S. Kim, H. Yin, J. Park, S. Ahn, B. Kang, K. Kim, G. Stefanovich, I. Song, and others, "Stack friendly all-oxide 3D RRAM using GaInZnO

peripheral TFT realized over glass substrates," in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, 2008, pp. 1–4.

[37] K. Szot, W. Speier, G. Bihlmayer, and R. Waser, "Switching the electrical resistance of individual dislocations in single-crystalline SrTiO3," *Nat. Mater.*, vol. 5, no. 4, pp. 312–320, 2006.

[38] M.-J. Lee, S. Han, S. H. Jeon, B. H. Park, B. S. Kang, S.-E. Ahn, K. H. Kim, C. B. Lee, C. J. Kim, I.-K. Yoo, and others, "Electrical manipulation of nanofilaments in transition-metal oxides for resistance-based memory," *Nano Lett.*, vol. 9, no. 4, pp. 1476–1481, 2009.

[39] W. Guan, S. Long, Q. Liu, M. Liu, and W. Wang, "Nonpolar Nonvolatile Resistive Switching in Cu Doped," *Electron Device Lett. IEEE*, vol. 29, no. 5, pp. 434–437, 2008.

[40] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[41] S. H. Jo, K.-H. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Lett.*, vol. 9, no. 1, pp. 496–500, 2008.

[42] D. B. Strukov and R. S. Williams, "Exponential ionic drift: fast switching and low volatility of thin-film memristors," *Appl. Phys. A*, vol. 94, no. 3, pp. 515–519, 2009.

[43] P. Sheridan, K.-H. Kim, S. Gaba, T. Chang, L. Chen, and W. Lu, "Device and SPICE modeling of RRAM devices," *Nanoscale*, vol. 3, no. 9, pp. 3833–3840, 2011.

[44] Z. Biolek, D. Biolek, and V. Biolkova, "SPICE model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.

[45] R. Sharpe and R. Palmer, "Evidence for field emission in electroformed metal-insulator-metal devices," *Thin Solid Films*, vol. 288, no. 1, pp. 164–170, 1996.

[46] R. Stratton, "Volt-current characteristics for tunneling through insulating films," *J. Phys. Chem. Solids*, vol. 23, no. 9, pp. 1177–1190, 1962.

[47] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, no. 13, pp. 5872–5878, 2013.

[48] B. Gao, S. Yu, N. Xu, L. Liu, B. Sun, X. Liu, R. Han, J. Kang, B. Yu, and Y. Wang, “Oxide-based RRAM switching mechanism: A new ion-transport-recombination model,” in *Electron Devices Meeting, 2008. IEDM 2008. IEEE International*, 2008, pp. 1–4.

[49] T. Chang, S.-H. Jo, K.-H. Kim, P. Sheridan, S. Gaba, and W. Lu, “Synaptic behaviors and modeling of a metal oxide memristive device,” *Appl. Phys. A*, vol. 102, no. 4, pp. 857–863, 2011.

[50] T. Chang, S.-H. Jo, and W. Lu, “Short-term memory to long-term memory transition in a nanoscale memristor,” *ACS Nano*, vol. 5, no. 9, pp. 7669–7676, 2011.

[51] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, and W. D. Lu, “Experimental Demonstration of a Second-Order Memristor and Its Ability to Biorealistically Implement Synaptic Plasticity,” *Nano Lett.*, vol. 15, no. 3, pp. 2203–2211, 2015.

[52] T. Kohonen, *Learning vector quantization*. Springer, 1995.

[53] T. Kohonen, G. Barna, and R. Chrisley, “Statistical pattern recognition with neural networks: Benchmarking studies,” in *Neural Networks, 1988., IEEE International Conference on*, 1988, pp. 61–68.

[54] T. Kohonen, “Improved versions of learning vector quantization,” in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, 1990, pp. 545–550.

[55] E. Oja, “Simplified neuron model as a principal component analyzer,” *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, 1982.

[56] T. Martinetz, K. Schulten, and others, *A“ neural-gas” network learns topologies*. University of Illinois at Urbana-Champaign, 1991.

[57] Y. LeCun, C. Cortes, and C. J. Burges, *The MNIST database of handwritten digits*. 1998.

[58] H. Lee, A. Battle, R. Raina, and A. Y. Ng, “Efficient sparse coding algorithms,” in *Advances in neural information processing systems*, 2006, pp. 801–808.

[59] Y. Singer and J. C. Duchi, “Efficient learning using forward-backward splitting,” in *Advances in Neural Information Processing Systems*, 2009, pp. 495–503.

[60] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, “Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition,” in

Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on, 1993, pp. 40–44.

[61] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Comput.*, vol. 20, no. 10, pp. 2526–2563, 2008.

[62] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?," *Vision Res.*, vol. 37, no. 23, pp. 3311–3325, 1997.

[63] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM Rev.*, vol. 43, no. 1, pp. 129–159, 2001.

[64] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[65] B. A. Olshausen and others, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[66] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online Dictionary Learning for Sparse Coding," in *Proceedings of the 26th Annual International Conference on Machine Learning*, New York, NY, USA, 2009, pp. 689–696.

[67] D. Salomon, *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[68] E. Candes and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Probl.*, vol. 23, no. 3, p. 969, 2007.

[69] M. Yang, L. Zhang, J. Yang, and D. Zhang, "Robust sparse coding for face recognition," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 625–632.

[70] D. Dai and W. Yang, "Satellite image classification via two-layer sparse coding with biased image representation," *Geosci. Remote Sens. Lett. IEEE*, vol. 8, no. 1, pp. 173–176, 2011.

[71] K. Yu, Y. Lin, and J. Lafferty, "Learning image representations from the pixel level via hierarchical sparse coding," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 1713–1720.

[72] C. Rozell, D. Johnson, R. Baraniuk, and B. Olshausen, "Locally Competitive Algorithms for Sparse Approximation," in *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, 2007, vol. 4, pp. IV – 169–IV – 172.

[73] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *ArXiv Prepr. Math0307152*, 2003.

[74] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imaging Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[75] J. M. Bioucas-Dias and M. A. Figueiredo, "A new TwIST: two-step iterative shrinkage/thresholding algorithms for image restoration," *Image Process. IEEE Trans. On*, vol. 16, no. 12, pp. 2992–3004, 2007.

[76] P. F. Schultz, D. M. Paiton, W. Lu, and G. T. Kenyon, "Replicating Kernels with a Short Stride Allows Sparse Reconstructions with Fewer Independent Kernels," *ArXiv Prepr. ArXiv14064205*, 2014.

[77] A. Hyvärinen and P. O. Hoyer, "A Two-Layer Sparse Coding Model Learns Simple and Complex Cell Receptive Fields and Topography From Natural Images," *Vision Res.*, vol. 41, no. 18, pp. 2413–2423, 2001.

[78] R. Mehrotra, K. R. Namuduri, and N. Ranganathan, "Gabor filter-based edge detection," *Pattern Recognit.*, vol. 25, no. 12, pp. 1479–1494, 1992.

[79] T. P. Weldon, W. E. Higgins, and D. F. Dunn, "Efficient Gabor filter design for texture segmentation," *Pattern Recognit.*, vol. 29, no. 12, pp. 2005–2015, 1996.

[80] V. Dragoi, J. Sharma, and M. Sur, "Adaptation-induced plasticity of orientation tuning in adult visual cortex," *Neuron*, vol. 28, no. 1, pp. 287–298, 2000.

[81] H.-Y. He, W. Hodos, and E. M. Quinlan, "Visual deprivation reactivates rapid ocular dominance plasticity in adult visual cortex," *J. Neurosci.*, vol. 26, no. 11, pp. 2951–2955, 2006.

[82] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano Lett.*, vol. 12, no. 1, pp. 389–395, 2011.

[83] T. Chang, “Tungsten oxide memristive devices for neuromorphic applications,” The University of Michigan, 2012.

[84] F. Alibart, E. Zamanidoost, and D. B. Strukov, “Pattern classification by memristive crossbar circuits using ex situ and in situ training,” *Nat. Commun.*, vol. 4, 2013.

[85] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, “A neuromorphic visual system using RRAM synaptic devices with Sub-pJ energy and tolerance to variability: Experimental characterization and large-scale modeling,” in *Electron Devices Meeting (IEDM), 2012 IEEE International*, 2012, pp. 10–4.

[86] S. Park, J. Noh, M. Choo, A. M. Sheri, M. Chang, Y.-B. Kim, C. J. Kim, M. Jeon, B.-G. Lee, B. H. Lee, and others, “Nanoscale RRAM-based synaptic electronics: toward a neuromorphic computing device,” *Nanotechnology*, vol. 24, no. 38, p. 384009, 2013.

[87] H. Choi, H. Jung, J. Lee, J. Yoon, J. Park, D. Seong, W. Lee, M. Hasan, G.-Y. Jung, and H. Hwang, “An electrically modifiable synapse array of resistive switching memory,” *Nanotechnology*, vol. 20, no. 34, p. 345201, 2009.

[88] I. Baek, D. Kim, M. Lee, H.-J. Kim, E. Yim, M. Lee, J. Lee, S. Ahn, S. Seo, J. Lee, and others, “Multi-layer cross-point binary oxide resistive memory (OxRRAM) for post-NAND storage application,” in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, 2005, pp. 750–753.

[89] M.-J. Lee, Y. Park, B.-S. Kang, S.-E. Ahn, C. Lee, K. Kim, W. Xianyu, G. Stefanovich, J.-H. Lee, S.-J. Chung, and others, “2-stack 1D-1R cross-point structure with oxide diodes as switch elements for high density resistance RAM applications,” in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, 2007, pp. 771–774.

[90] J. D. Plummer, *Silicon VLSI technology: fundamentals, practice, and modeling*. Pearson Education India, 2000.

[91] W. Whyte, *Cleanroom technology: fundamentals of design, testing and operation*. John Wiley & Sons, 2010.

[92] T. Chang, P. Sheridan, and W. Lu, “Modeling and implementation of oxide memristors for neuromorphic applications,” in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, 2012.

[93] P. Sheridan, W. Ma, and W. Lu, “Pattern recognition with memristor networks,” in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, pp. 1078–1081.

[94] S. Choi, P. Sheridan, and W. D. Lu, “Data Clustering using Memristor Networks,” *Sci. Rep.*, vol. 5, 2015.

[95] P. Sheridan, C. Du, and W. Lu, “Feature Extraction Using Memristor Networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, 2015.