11-2015

Design of Neuromemristive Systems for Visual Information Processing

Cory E. Merkel
cem1103@rit.edu

R·I·T

Design of Neuromemristive Systems for Visual Information Processing

by

Cory E. Merkel

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Microsystems Engineering

Microsystems Engineering Program
Kate Gleason College of Engineering

Rochester Institute of Technology
Rochester, New York
November 2015

Design of Neuromemristive Systems for Visual Information Processing
By
Cory E. Merkel

Committee Approval:

We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Microsystems Engineering.

Certified By:

ABSTRACT

Kate Gleason College of Engineering
Rochester Institute of Technology

Degree Doctor of Philosophy **Program** Microsystems Engineering
Author's Name Cory E. Merkel
Advisor's Name Dhireesha Kudithipudi, Ph.D.
Dissertation Title Design of Neuromemristive Systems for Visual Information Processing

Neuromemristive systems (NMSs) are brain-inspired, adaptive computer architectures based on emerging resistive memory technology (memristors). NMSs adopt a mixed-signal design approach with closely-coupled memory and processing, resulting in high area and energy efficiencies. Previous work suggests that NMSs could even supplant conventional architectures in niche application domains such as visual information processing. However, given the infancy of the field, there are still several obstacles impeding the transition of these systems from theory to practice.

This dissertation advances the state of NMS research by addressing open design problems spanning circuit, architecture, and system levels. Novel synapse, neuron, and plasticity circuits are designed to reduce NMSs' area and power consumption by using current-mode design techniques and exploiting device variability. Circuits are designed in a 45 nm CMOS process with memristor models based on multilevel (W/Ag-chalcogenide/W) and bistable (Ag/GeS_2/W) device data. Higher-level behavioral, power, area, and variability models are ported into MATLAB to accelerate the overall simulation time. The circuits designed in this work are integrated into neural network architectures for visual information processing tasks, including feature detection, clustering, and classification. Networks in the NMSs are trained with novel stochastic learning algorithms that achieve $\approx 3.5\times$ reduction in circuit area, reduced design complexity, and exhibit similar convergence properties compared to the least-mean-squares algorithm. This work also examines the effects of device-level variations on NMS performance, which has received limited attention in previous work. The impact of device variations is reduced with a partial on-chip training methodology that enables NMSs to be configured with relatively sophisticated algorithms (e.g. resilient backpropagation), while maximizing their area-accuracy tradeoff.

Contents

List of Figures

List of Tables

Frequently Used Symbols[1,2]

a	Mean distance between defects in a memristor
A	Circuit area
A_0	Open-loop gain
α	NMS learning rate
A_v	Voltage gain
$A_{V_{th0}}$	Fitting parameter for MOSFET V_{th0} mismatch
B	Boost factor
β	Geometry-dependent MOSFET current factor
C	Capacitance or a binary number to be represented stochastically
χ	Function that governs the change in memristor state variable
C_{ox}	MOSFET oxide capacitance
C_s	Stochastic bit stream or bundle
d	Distance measure for clustering
D	Memristor film thickness or duty factor
δ	Variation operator
$\delta(\cdot)$	Dirac delta function
E	Total energy
E_a	Activation energy
η	Activity factor (e.g. neuron activity)
f	Neuron activation function
F	Wire pitch or folding amplifier folding factor or electric field
g	Memristor conductance ratio
G	Conductance
γ	Memristor state variable
g_m	Transconductance
G_m	Memristor conductance
G_{mon}	Memristor *on* conductance
G_{moff}	Memristor *off* conductance
h	NMS hypothesis function
$\hbar$	Planck's constant divided by 2π

[1]In general, the symbols defined here may contain superscripts or subscripts to specify indices, component names, maximum, minimum, average, etc.

[2]All symbols listed here are scalar values (e.g. x), but in general, they may be components of vectors, denoted by lowercase boldface symbols (e.g. $\mathbf{x}$) or matrices, denoted by uppercase boldface symbols (e.g. $\mathbf{X}$).

$H(\cdot)$	Heaviside step function
i_{ds}	MOSFET drain current
i_{ds0}	Constant prefactor in the MOSFET long channel drain current
i'_{ds}	Long channel MOSFET drain current
i_m	Memristor current
I_{max}	Constant current value used for normalization in an NMS
L	MOSFET channel length
Λ	Multiplicative effect of drain-source voltage on MOSFET drain current
lr	Number of clock cycles to apply memristor write voltage
m	Number of training examples or mass
μ_0	MOSFET low-field mobility
n	MOSFET subthreshold slope factor
N	Number of NMS inputs or length (width) of a stochastic bit stream (bundle)
N_{epochs}	Number of training epochs
N_h	Number of hidden-layer neurons in an NMS
N_w	Number of weights/synapses in an NMS
N_x	Number of neurons in an NMS
ν	Attempt-to-escape frequency
o	NMS input dimensionality after dimensionality reduction
O	NMS input dimensionality before dimensionality reduction
p	NMS training/test vector index or momentum
P	Power consumption
ϕ	Flux linkage
π	Reduced dimensionality SLMS input vectors
Π	Ratio of Λ values for MOSFETs in a current mirror
$\Pr(\cdot)$	Probability
p_{switch}	Memristor switching probability
ψ	Quantum wave function
Ψ	Number of bits in a binary number to be represented stochastically
q	Elementary electric charge
R	Resistance
R_{in}	Circuit input impedance
R_m	Memristor resistance
r_o	MOSFET output impedance
R_o	Circuit output impedance
Θ	Factor related to saturated MOSFET drain current on v_{ds} in subthreshold
s	Synaptic output or sum of synaptic outputs
T	Temperature or tunneling probability or period
θ	Threshold for threshold activation function
ts	Transistor size factor
t_w	Memristor write time
u	NMS input
U	Stochastic NMS input or potential energy
v_{ds}	MOSFET drain-source voltage
v_{gs}	MOSFET gate-source voltage

v_m	Memristor voltage
V_L	Lower bound of gate-source voltage for MOSFET weak inversion
V_M	Upper bound of gate-source voltage for MOSFET weak inversion
V_{DD}	Positive supply voltage
$V_{ds_{sat}}$	MOSFET saturation voltage
V_{sb}	MOSFET source-body voltage
V_{SS}	Negative supply voltage
V_{tn}	Negative memristor threshold voltage
V_{tp}	Positive memristor threshold voltage
V_{th0}	MOSFET threshold voltage
V_{th}	Folding amplifier neuron threshold voltage
V_T	Thermal voltage
v_w	Memristor write voltage
w	Synaptic weight
w'	Synaptic weight without effect of v_{ds}
W	MOSFET channel width
x	Neuron output (activation)
X	Stochastic neuron output (activation)
ξ	Memristor model fitting parameter
y	Expected NMS output (target)
Y	Stochastic expected NMS output (target)
$\hat{y}$	NMS output
$\hat{Y}$	Stochastic NMS output
ζ	MOSFET model fitting parameter

To my beautiful wife, Melissa,
for all of your patience, love, and encouragement.

Acknowledgements

This research was made possible by the support and encouragement of a number of people inside and outside of RIT. First, I'd like to acknowledge the Air Force Research Laboratory and National Science Foundation (NSF EAGER Grant ECCS-1445386) for partially funding this research. In addition, I'd like to thank my advisor, Dr. Dhireesha Kudithipudi, who has been a wonderful mentor and role model during the course of my Ph.D. work. I am also grateful for the support and flexibility of my dissertation committee: Dr. Ray Ptucha, Dr. Santosh Kurinec, Dr. Bryant Wysocki, Dr. Brad Mahon, Dr. Haibo He, Dr. Manan Suri, and Dr. Gabrielle Gaustad.

The RIT Computer Engineering Department's faculty and staff, including Dr. Shanchieh Yang, Dr. Andres Savakis, Kathy Stefanic, Pam Steinkirchner, Anne DeFelice, Rick Tolleson, Emilio Del Plato, and Sarah Buell, have been extremely helpful with administrative tasks, providing resources, and addressing technical problems. I also want to thank the faculty and staff of the Microsystems Engineering Ph.D. Program, including Dr. Bruce Smith and Lisa Zimmerman.

Several researchers from the Air Force Research Laboratory, including Dr. Bryant Wysocki, Dr. Garrett Rose, Nathan McDonald, and Claire Thiem have been outstanding, providing valuable technical contributions to this work. Additional collaborators include Dr. Nate Cady (SUNY Albany), Dr. Kris Campbell (Boise State University), and Alex Nugent (Knowm, Inc.). I must also thank all of the members of the NanoComputing Research Lab, including Dan Christiani, James Mnatzaganian, Qutaiba Saleh, Abdullah Zyarah, James Thesing, and many more. They have been a wonderful source of encouragement and technical insight. In addition, thank you to the many friends in my undergraduate class at RIT, including David Brenner, Mike Sanfilippo, Daniel Liu, Jeff Kemp, and many more.

Finally, a sincere and heartfelt thank you to my wife, mom, dad, brother, and the rest of my family and friends for all of their support.

Chapter 1

Introduction

How can emerging nanotechnologies be exploited to design the next generation of intelligent computers? This is the central question explored in this dissertation and the underlying theme of neuromemristive systems (NMSs). An NMS is a brain-inspired, special-purpose computing platform based on nanoscale resistive memory (memristor) technology. NMSs represent a subclass of a broader movement in brain-inspired computing called neuromorphic systems, which were pioneered by Carver Mead in the late 1980s [1]. The primary goal of both neuromorphic and neuromemristive systems is to provide levels of intelligent information processing, adaptation/learning, energy/area efficiency, and noise/fault tolerance in niche application domains that are not achievable using conventional computing paradigms. Conventional computer architectures are limited in these aspects because of their adherence to the von Neumann model, where the hardware is digital and immutable, computation is sequential and precise, and a distinct separation exists between computation and memory. Although the von Neumann model is unparalleled for well-defined sequential problems (e.g. arithmetic and logic), it is ill-suited in application domains such as visual information processing, where problems are not well-posed, data are analog and noisy, and solutions are inherently parallel. Mead and many researchers before him recognized that biological systems such as the primate brain solve these types of problems with much greater efficiency than conventional computing systems. In fact, it is estimated that for applications such as visual information processing, the brain is a factor 1×10^7 more energy efficient than any conceivable digital computer. The explanation for this large efficiency gap lies in the stark contrast between conventional computer architectures and the computing methods employed by the brain.

The human brain is inherently mixed-signal, massively parallel, approximate, and plastic, giving rise to its incredible processing ability, low power consumption, and capacity for adaptation. Both neuromorphic and neuromemristive systems attempt to emulate brain functionality with neural networks built from mixed-signal circuits. The two distinguishing features of an NMS are:

- The incorporation of memristive devices into NMSs enables plasticity at multiple levels, beyond the synaptic plasticity that is typically implemented in neuromorphic systems.

- NMSs focus on abstraction of the computational principles found in the nervous system rather than biological plausibility. This approach is better for two reasons. First, it is still unclear how behavior at the level of single neurons and small neuronal populations leads to system-level behavior of the brain. Second, the basic components of the brain (e.g. proteins, cells, etc.) are much different than those used in integrated circuit (IC) design (e.g. transistors and memristors). Therefore, it is unlikely that copying the brain's structure in an IC will yield the same emergent properties.

Note that there are other computing platforms that are attractive for brain-like information processing, including general-purpose graphical processing units (GPGPUs) and field-programmable gate arrays (FPGAs). GPGPUs are optimized for the types of linear algebra computations that govern neural network behavior. However, they lack the reconfigurability that is offered by NMSs. On the other hand, FPGAs have a high degree of reconfigurability, but they have very high area and power overheads to support their interface and routing resources.

Neuromorphic and neuromemristive systems have been attracting a lot of research attention through various projects, such as the DARPA SYNAPSE [2] program, DARPA's Cortical Processor Program [3], Physical Intelligence [4], UPSIDE [5], the Human Brain Project [6], and the Blue Brain Project [7]. Vision-related applications have been the targets of many of these projects and other programs. Visual information processing has a large application spectrum, including surveillance, medical imaging, content filtering/searching, object classification, and many others. In order to explore these applications within NMSs, one must first consider the breadth of the NMS design space, which spans circuits, architectures, and system-level components. Although NMSs have the potential to replace von Neumann architectures in niche application domains, there are several gaps in existing work, particularly at the circuit level, that have impeded their transition from research to practice. This dissertation fills those gaps, exploring the design and modeling of primitive circuits and learning algorithms to improve the efficiency and variation tolerance of visual information processing in NMSs. The specific novel contributions of this work are:

- New current-mode synapse, neuron, and plasticity circuits for NMSs that have reduced area and power consumption over voltage-mode designs.
- Novel stochastic training algorithms for NMSs with circuit implementations that have reduced area overhead and complexity.
- Integration of the circuits and training algorithms designed this work into NMSs for visual feature detection, clustering, and classification.
- Detailed models and analyses of the effects of device-level variations on system-level accuracy, enabling the development of better training methodologies.

The rest of this dissertation is outlined as follows: Chapter 2 provides background and related work on the human visual system, memristor devices, NMSs, and stochastic computation. Chapter 3 presents the metal-oxide-semiconductor field-effect transistor (MOSFET) and memristor models used in this work, the strategy for circuit and system-level simulations, and the current-mode design methodology adopted in this research. Chapter 4 presents the synapse and neuron circuits designed in this work, along with models of their area, power consumption, and variation analyses. Novel NMS training algorithms and circuit-level implementations are presented in Chapter 5. Chapter 6 demonstrates the utility of the circuits and training algorithms designed in this work for visual information processing tasks. The effects of device-level variations on system-level performance are evaluated in Chapter 7. Chapter 8 concludes this dissertation.

Chapter 2

Background and Related Work

2.1 Overview of the Human Visual System

The human visual system and visual perception are remarkable products of evolution. We can easily classify objects, identify emotions from facial cues, approximate distances, understand scenes, and perform several other complex visual processing tasks. A simplified depiction of the human visual pathways is shown in Figure 2.1. Light enters the eyes, stimulating photoreceptors (rods and cones) on the retina. The response of the stimulus propagates to ganglion cells whose axons leave the eye via the optic nerve. The eyes actually act as a pre-processor for visual information by transforming the single-ended input signal to a differential signal, making it noise-tolerant and invariant to overall light level. After information leaves the eyes, it follows the optic nerves and eventually the optic tract to the lateral geniculate nucleus (LGN) of the thalamus. From there, information radiates through several pathways in the temporal and parietal lobes collectively called Meyer's loop, eventually reaching the V1 and other visual cortices in the occipital lobe. It is here that several low-level features are extracted, such as edges, corners, etc. The pathway from the retina to the V1 cortex is known as the retino-geniculo-striate pathway. From the occipital lobe, visual information takes two distinct pathways to the parietal (dorsal stream) and temporal (ventral stream) lobes. Processing in these locations allows us to locate objects in space/time, and identify/classify objects. This work will focus on the ventral stream pathway, where neurons in the inferotemporal (IT) cortex respond to classes of objects. Higher-level cortical areas such as the prefrontal cortex (PFC) perform clustering operations, where we can associate images with similar meaning.

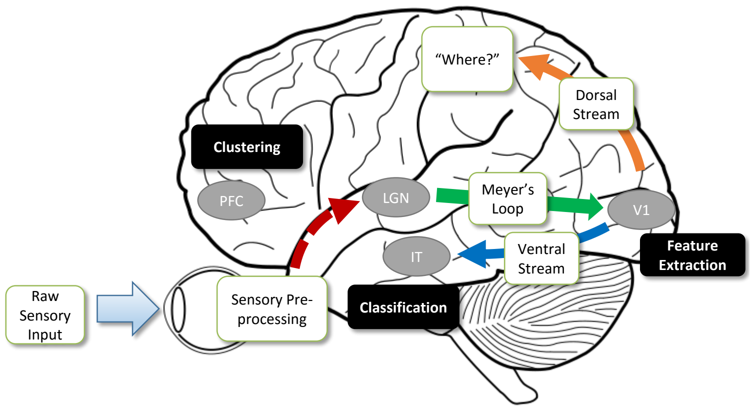


Figure 2.1: Lateral view of the human brain's visual pathways. Black filled rectangles show the visual information processing tasks explored in this work.

2.2 Memristors for Plasticity in Neuromemristive Systems

Our abilities to learn a new face, drive a car, identify objects, and perform other visual tasks are the results of brain plasticity. Plasticity is the characteristic of a system that allows it to undergo permanent changes in response to an external force. Biological systems exhibit remarkable levels of plasticity, enabling organisms to adapt to a changing environment, maintain a homeostatic state, and recover from injury. The same characteristics are of interest for future computing systems as they facilitate reconfigurability and noise tolerance,

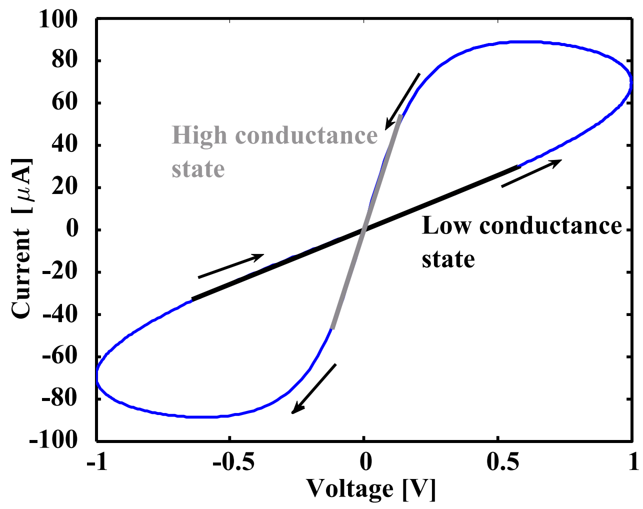


Figure 2.2: Pinched hysteresis current-voltage relationship that characterizes memristive devices.

reliability, and self-healing/resilience. The mechanisms that enable plasticity in a biological context occur at multiple scales, from the level of individual cells up to functional brain regions. These include neurogenesis, epigenetic mechanisms, long-term potentiation and depression in chemical synapses, and changes in topological mappings between brain regions and brain functions (e.g. retinotopic maps). At an abstract level, each of these plasticity mechanisms requires some form of memory. In particular, there is a certain level of persistence in e.g. the locations of specific neurons, the efficacy of synaptic transmission in a particular synapse, and the topology of brain regions. Hence, any brain-inspired computing system should ideally employ some form of non-volatile memory (NVM) to achieve plastic behavior.

Flash has been the dominant non-volatile memory technology used in computing systems for many years because of its high density and low cost. However, due to many scaling-related challenges, flash is expected to be superseded by a novel memory technology within the next decade. Table 2.1 shows a comparison of NAND flash and prototypical/emerging non-volatile memories across energy, performance, and reliability metrics. Biologically-motivated targets for each metric are listed in the right column. In particular, phase change memory (PCM), spin transfer torque random access memory (STT-RAM), and resistive random access memory (RRAM) are among the most promising candidates for future NVM implementations [8]. Each of these technologies may also be described as a memristor or memristive device. A memristor is a two-terminal passive circuit element that follows a state-dependent Ohm's Law, characterized by a pinched hysteresis current-voltage relationship as shown in Figure 2.2 [9–11]:

$$i_m(t) = G_m(\gamma)v_m(t) \tag{2.1}$$

$$\frac{\mathrm{d}\gamma}{\mathrm{d}t} = \chi(\gamma, v_m(t)) \tag{2.2}$$

where i_m is the current through the memristor, v_m is the voltage across the memristor, $\gamma \in [0, 1]$ is a state variable, $G_m(\gamma)$ is the state-dependent conductance, and χ governs how γ changes over time. The conductance will range from $G_{moff} \equiv G_m(\gamma = 0)$ to $G_{mon} \equiv G_m(\gamma = 1)$. By applying short voltage pulses to these devices, one can incrementally modify their conductance states, enabling the storage of multi-level memory values.

The most important metrics for an NVM technology within an NMS are dynamic range, number of memory states, retention, energy efficiency, and endurance. A memristor's dynamic range can be measured as the ratio of its *on* and *off* conductances (G_{mon}/G_{moff}).

Table 2.1: Comparison of non-volatile memory technologies for brain-inspired computing [8, 12–14].

		Memristors			
Metric	**Flash**	**PCM**	**STT-RAM**	**RRAM**	**Targets**
Dynamic Range $(\mho/\mho)$	-	>1000	2	1000	>4
Number of States	8-16	100	4	100	20-100
Retention	Several years at room temp.				years
Energy (pJ/bit)	>100	2-25	0.1-2.5	0.1-3	0.01
Endurance (cycles)	10^4	10^9	10^{15}	10^{12}	10^9

A large dynamic range allows sense circuitry to easily distinguish an NVM cell's different memory states. The number of states that the NVM cell can achieve has a direct impact on the area and energy efficiencies, as well as the functionality of an NMS. The number of memory states in a memristive device is equivalent to the number of distinguishable $G_m(\gamma)$ values that exist. For two conductance states to be distinguishable, they need to yield two different current levels (when placed in a circuit) that have a range which is larger than the noise level (e.g. thermal and shot noise) of the circuit. It may take several bi-stable (only able to achieve two memory states) NVM cells in an NMS to attain the same level of functionality as an NMS with a single NVM cell that has many memory states. Retention is another critical characteristic for an NVM technology. Within an NMS, a large retention allows the system to accumulate and integrate information over long periods of time. Low power is a primary NMS design goal, making energy-per-bit a critical metric in evaluating NVM technologies within these systems. Finally, in order for an NMS to learn and adapt, its underlying memory must be able to endure a large number of write events. Based on these metrics, RRAM is the most suitable NVM for NMS implementation. Although it has a good dynamic range, number of states, and retention, PCM requires high energy and voltages for writing. In addition, its endurance is borderline. In contrast, STT-RAM has very high endurance, but its dynamic range and number of resistance states are too small for NMS implementation. In addition, RRAM has excellent compatibility with CMOS and is highly scalable; its competition with other emerging NVM technologies will continue to fuel research that will be fruitful for RRAM-based NMSs.

RRAM cells, which will be referred to as memristors for the rest of this document, have a metal-insulator-metal (MIM) structure, where two conducting electrodes sandwich a thin-film switching layer. Various MIM memristor stacks have been explored, and there are several ways to categorize them based on their material properties (e.g. crystalline structure, band gap, etc.), proposed switching mechanism (e.g. anion, cation, Ferroelectric, etc.), or observed switching characteristics (e.g. bipolar or unipolar switching). Comprehensive reviews are provided in [12, 15–18]. The proposed switching mechanism for most of the fabricated devices is based on redox reactions and migration of defects such as interstitial ions or vacancies. Several different models have been proposed to capture the physical phenomena underlying memristive behavior [18–21]. However, many of them are computationally expensive and are not amenable to large-scale circuit simulations. Simpler empirical models such as the PWL model proposed in [22] are parametrized by experimental memristor data and have lower computational complexity. Finally, several groups have proposed simulation program with integrated circuit emphasis (SPICE) or Verilog AMS models for circuit-level simulations [23–30].

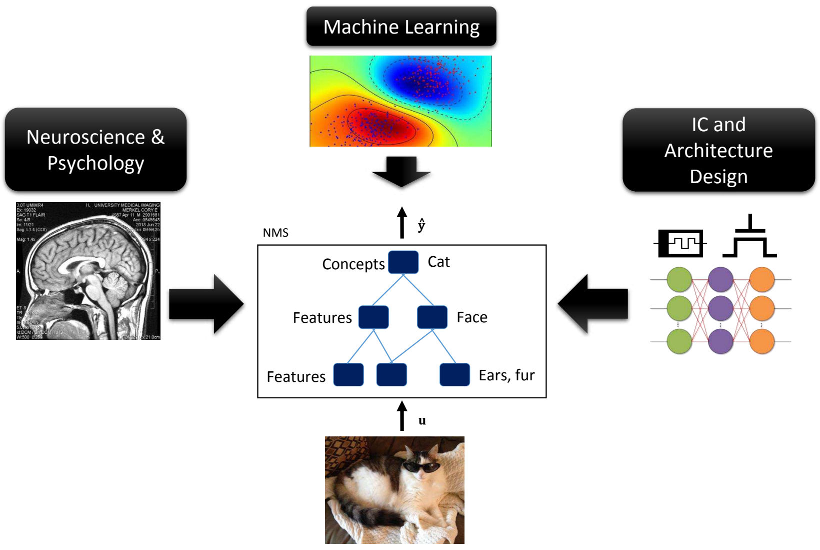


Figure 2.3: High-level depiction of an NMS. The NMS design process requires interdisciplinary collaboration between experts in neuroscience/neuropsychology, machine learning, and integrated circuit/architecture design. In this particular example, an NMS is designed for image classification. A neural network architecture is used to extract features at lower levels and make predictions at higher levels.

2.3 Neuromemristive Systems

NMSs leverage memristors' small footprint, simple structure, potential for high density (possibly on the order of 10^{14} bits/cm^2 [31]), and capacity for incremental multi-level memory to achieve plastic behavior. Figure 2.3 presents a high-level depiction of an NMS. Here, it is emphasized that NMS designs are inspired by principles from the neuroscience/neuropsychology, machine learning, and IC design domains. An NMS has three levels of design abstraction. Namely, there are primitive (circuit), architectural, and system-level designs. The NMS design space is shown in Figure 2.5. At the circuit, or primitive level, choices related to memristive devices, signaling type (e.g. analog or digital), mode, and interface between different technologies are important. Synaptic weighting circuits provide a weighted connections between neurons: $s_{i,j} = x_j w_{i,j}$, where $s_{i,j}$ is the synapse's output, $w_{i,j}$ is the weight of the connection, and x_j is the input to the synapse. Neuron circuits sum their inputs s_i and apply an activation function f to produce an output x_i. Finally, synaptic adaptation circuits are used to modify the synaptic weights by changing the conductance states of their memristors. At the architecture level, a topology and a training algorithm must be specified. Topologies often take the form of neural network structures such as multilayer perceptrons (MLPs), and training algorithms range from simple unsupervised learning rules to complex algorithms such as backpropagation. At the system level, multiple architectures can be combined and applied to a specific problem. In Figure 2.3, the NMS is analyzing a picture to determine the most-likely classification given the pictures that it has seen before. More generally, an NMS maps an input vector $\mathbf{u}$ to an output vector $\hat{\mathbf{y}}$ through a hypothesis function h. Other design choices related to data pre-processing, classification/regression algorithms, and partitioning of functions across different parts of the system must also be made.

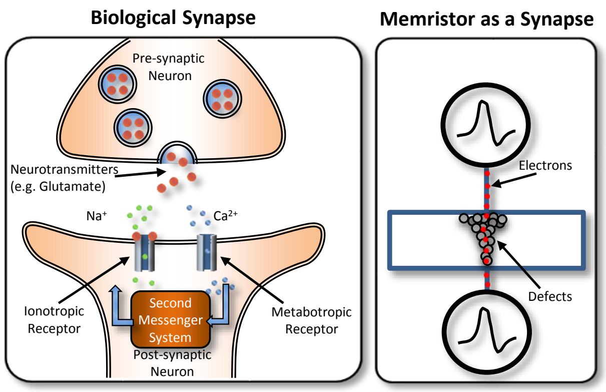


Figure 2.4: Comparison of a biological synapse and a memristor as a synapse emulator.

There is an abstract behavioral similarity between biological synapses and memristors which has sparked wide interest in the use of these devices as hardware synapses. A memristive synapse in an NMS must provide three functions: point-to-point communication,

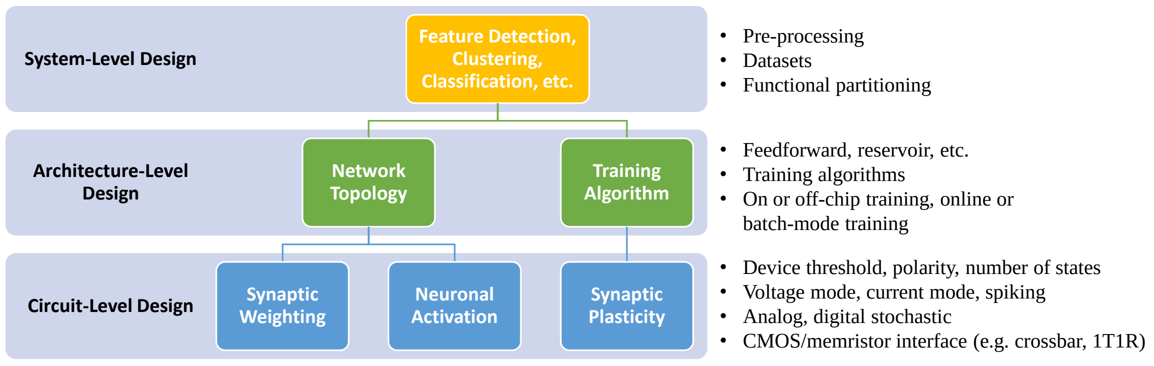

System-Level Design
Feature Detection, Clustering, Classification, etc.
Pre-processing
Datasets
Functional partitioning
Architecture-Level Design
Network Topology
Training Algorithm
Feedforward, reservoir, etc.
Training algorithms
On or off-chip training, online or batch-mode training
Circuit-Level Design
Synaptic Weighting
Neuronal Activation
Synaptic Plasticity
Device threshold, polarity, number of states
Voltage mode, current mode, spiking
Analog, digital stochastic
CMOS/memristor interface (e.g. crossbar, 1T1R)

Figure 2.5: NMS design space for visual information processing.

linear computation[1], and learning. In theory, a single memristor is sufficient to provide all three synaptic functions [32]. This idea is illustrated in Figure 2.4. In the case of a biological synapse, information is communicated between a pre-synaptic and post-synaptic neuron through the diffusion of neurotransmitter molecules. For example, the neurotransmitter glutamate opens ligand-gated ion channels at the post-synaptic neuron's dendrite allowing charge-carrying ions to diffuse in. The strength, or weight of this communication pathway is dependent on the number of ion channels present and the efficacy of each channel in facilitating ion diffusion. Furthermore, the synaptic weight can be changed by adding or removing ion channels or changing their transmission efficacy. In comparison, memristive devices can communicate information between their electrodes via electrons. The weight of transmission depends on the state of the memristive device. The weight can be changed by modifying the memristor's defect state.

Several groups have leveraged the similarity illustrated in Figure 2.4 in networks of spiking neurons that implement spike time-dependent plasticity (STDP)-based Hebbian/anti-Hebbian learning. Networks of analog spiking neurons with single-memristor synapses are

[1]The synaptic computation can be non-linear as well, but linear computation is better for implementation of most neural network architectures.

presented in [33, 34], and a digital implementation is proposed in [35]. However, these implementations require neurons to output three different voltage levels, complicating the hardware neuron design. Furthermore, the digital implementation requires a complex spiking sequence controlled by a finite state machine. Single memristors have also been used for binary synapse (*on* and *off* states only) realization in cellular neural networks [36]. However, additional circuitry is generally needed in a memristor-based synapse design (switches, current mirrors, etc.) depending on which type(s) of learning algorithms (e.g. supervised or unsupervised learning, synchronous or asynchronous learning, etc.) and neuron designs (e.g. neuron activation function, analog/digital implementation, etc.) are present in the network. In [37], a series combination of an ambipolar thin-film transistor (TFT) and a memristor is used for synaptic transmission and weight storage in a spiking neural network. The gate of the TFT is controlled by the pre-synaptic neuron, enabling or disabling a constant voltage to pass through the memristor, creating a memristance-modulated current at the input of the post-synaptic neuron. The authors demonstrate learning in a two-neuron network with an average-spike frequency-based learning rule. Kim et al. [38, 39] propose a memristor synapse based on a bridge circuit and a differential amplifier. It can be programmed to implement both positive (excitatory) and negative (inhibitory) weights. It also has good noise performance due to its fully differential architecture. However, it requires 3 MOSFETs, 5 memristors, and additional training circuitry (depending on which learning algorithm is being implemented). Another synapse design, presented in [40], incorporates two memristors which can be trained to provide a desired ratio of excitation to inhibition. The synapse also allows bidirectional communication. However, the authors do not include a detailed description of the training circuitry, and the design also consumes a constant static power, which could cause high power dissipation in large networks. Another memristor-based synapse design is proposed in [41]. The design operates in subthreshold, resulting in low power consumption. However, the charge sharing technique that the authors employ requires separate pre-charge and evaluate phases of operation, similar to dynamic logic.

At the architecture and system levels, NMSs have been designed for associative memory [42], brain-state-in-a-box recall [43], temporal pattern recognition in a reservoir network [44], implication logic [36], and RRAM architectures [17, 45–50]. This work focuses on vision because of its numerous application domains and its well-established models within the brain. Other groups have designed NMSs for vision-related applications. In [35] a neuromemristive winner-take-all type network is designed to detect the position of an object. STDP is used for unsupervised training. The authors of [51] propose the integration of a memristor bridge synapse into a multilayer perceptron network for classifying automobiles. An NMS for optical character recognition (OCR) is designed in [52] using a simple feedforward network and STDP training. In [53], the authors propose an NMS that uses stochastic conductive bridge random access memory (CBRAM) devices for visual pattern extraction.

2.4 Stochastic Computation

One of the contributions of this work is the design of an NMS training algorithm based on stochastic logic. The roots of stochastic computing can be traced back to the work of John von Neumann. In his 1956 paper [54], von Neumann proposes using a bundle of N wires to send a message. If $(1 - \triangle)N$ or more wires carry a 1, then the message is interpreted as a 1, where $0 \leq \triangle \leq 1/2$. If $\triangle N$ or fewer wires carry a 1, then the message is interpreted as a 0. The idea of distributing a message across multiple wires or multiple time windows on a single wire was attractive to the machine learning community. Indeed, stochastic computing grew out of the need to reduce hardware complexity, power, and unreliability in machine learning applications [55, 56]. Stochastic computing achieves these goals simply by changing the way data is represented in a computer.

2.4.1 Stochastic Representation of a Digital Value

At the heart of stochastic computing is the stochastic representation of digital values. In a unipolar [56] stochastic representation, an Ψ-bit number $C \in \{0, 1, \dots, 2^{\Psi} - 1\}$ is mapped to an N-bit stream (serial) or bundle (parallel) $C_s = C_{s_{N-1}}, C_{s_{N-2}}, \dots, C_{s_0}$. Then, C_s is interpreted as the probability that any C_{s_i} will be a logic 1. For example, the stream or bundle 0,1,0,1,0,0 represents the probability 2/6. The stream or bundle is characterized by a Bernoulli process $X = X_{N-1}, X_{N-2}, \dots, X_0$, where [57, 58]

$$x \equiv \Pr(X = 1) \equiv \Pr(X_i = 1) = 1 - \Pr(X_i = 0) = \frac{C}{2^{\Psi} - 1} \tag{2.3}$$

Here, $X_i = 0$ if $C_{s_i} = 0$, or 1 otherwise. It is also possible to have a bipolar stochastic representation such that $X_i = -1$ if $C_{s_i} = 0$ or 1 otherwise. In this case, x is defined as [56]

$$x \equiv 2\Pr(X = 1) - 1 \tag{2.4}$$

The bipolar representation is useful in the cases where negative numbers need to be represented.

Converting from the digital to the stochastic representation can be achieved using a random number generator and a comparator. If the random number is less than or equal to the value held in a register, then a logic 1 value is produced on the output. Otherwise, the comparator output is logic 0 [57]. As N becomes large, $\Pr(X = 1)$ approaches the value in (2.3). Linear feedback shift registers (LFSRs) are commonly used to generate pseudorandom numbers. Note that the bits in C_s could also be generated in parallel, but this would require N independent random number generators and N comparators. Converting from a stochastic representation back to a digital number can be achieved by counting the number of 1s in the stochastic bit stream. In this work, analog values, rather than digital values, are converted to stochastic bit streams, requiring different circuitry.

One advantage of the stochastic representation is its inherent fault tolerance. Consider the binary representation of an $\Psi + 1$-bit binary number $C = C_\Psi C_{\Psi-1} \ldots C_0$. In the case of a single soft error, one bit of C is flipped, producing a new number C'. The maximum error occurs when C_Ψ is flipped. In that case, the error is $|C - C'| = 2^\Psi$. However, when the same number is represented stochastically, the maximum error is only $|C_s - C_s'| = 1$.

2.4.2 Stochastic Arithmetic Operations

Stochastic data representation has another key advantage, where arithmetic operations such as multiplication become trivial to implement in hardware. For example, consider a 2-input AND gate with inputs mapped to stochastic bit streams X^1 and X^2. Let the probabilities (probability that a 1 will occur) of the bit streams be $x_1 = 3/6$ and $x_2 = 2/6$. The output probability is $y \equiv \Pr(Y = 1) \equiv \Pr(X^1 = 1 \text{ and } X^2 = 1)$. If X^1 and X^2 are statistically independent, then $y = x_1 x_2 = 1/6$ [59]. Therefore, a single AND gate can be used to multiply two numbers in the stochastic domain. In general, any logic function $g = f(I_1, I_2, \ldots, I_n)$ with inputs mapped to independent stochastic bit streams $X^1, X^2, \ldots, X^n$ will have an output probability of [59]

$$y \equiv \Pr(Y = 1) \equiv \sum_{I_1,\ldots,I_n : f(I_1,\ldots,I_n)=1} \left(\prod_{k=1}^{n} \Pr(X^k = I_k) \right), \tag{2.5}$$

which is a multivariate polynomial in $x_1, x_2, \ldots, x_n$ with integer coefficients and powers no greater than 1. Integration, division, square root, and squaring operations have also been demonstrated using stochastic logic [56, 57]. In [56], a state machine-based stochastic logic architecture is used to implement more complex functions such as the hyperbolic tangent.

Several applications and examples of stochastic computing have been reported in the literature. In [60], stochastic logic with parallel bit streams (bundles) is used to synthesize logic on self-assembled nanowire crossbar arrays. A neurochip based on stochastic logic is designed and fabricated in [61]. Overall, stochastic computing has several advantages over deterministic computation. These include reduced hardware complexity, fault tolerance, single-wire communication of signals, easy implementation of pipelining, and the ability to trade off performance and accuracy [56].

2.5 Summary

This chapter reviewed background and existing work related to NMSs. The key topics/ideas covered in this chapter were

- The human visual system is composed of feedforward pathways with areas dedicated to feature detection, clustering, and classification. Well-studied pathways in the brain, such as the retino-geniculo-striate and ventral stream pathways can be modeled using neural networks.

- The brain's visual system is plastic, allowing us to make predictions about what we see based on our past experience. This plasticity occurs at many levels of abstraction, such as in the adaptive strength of synapses.

- Memristors have a behavioral similarity to biological synapses and have been used in previous NMS designs to facilitate learning.

- Stochastic representation of data can reduce the hardware complexity of an NMS.

- Key aspects that are missing or have received limited attention from previous work include exploration of current-mode circuit designs, circuit design based on realistic (experimentally-driven) memristor models, exploration of non-monotonic neuronal activation functions, reduced complexity stochastic training algorithms, exploration of hardware-friendly topologies (e.g. single-layer networks), and the effect of device variations on system-level performance.

Chapter 3

Device Models, Simulation Strategy, and Design Methodology

This chapter discusses the MOSFET and memristor device models, as well as the simulation strategy and circuit design methodology used in this work. A 45 nm low power predictive technology MOSFET model (PTM) was used for all of the circuit designs presented in this research. The PTM model and a simplified behavioral MOSFET model are discussed in Section 3.1. Section 3.2 discusses the memristor models used in this work. Section 3.3 presents the simulation/verification strategy used in this research. Section 3.4 concludes this chapter with an overview of the simulation/verification strategy that was adopted for circuit and system-level

3.1 45 nm Low Power PTM MOSFET Characterization

The MOSFET model chosen for this work is a 45 nm high threshold (low power) PTM model. The detailed device behavior is captured using Berkeley's BSIM4.0 (SPICE model level 54). The BSIM4.0 model accounts for short channel effects, narrow width effects, non-uniform doping, mobility degradation, velocity saturation, and many other non-ideal behaviors. While this is good for verification purposes, the model becomes too complex for system-level design and analysis. Therefore, a simplified model is developed to capture the essential device behavior in a more tractable set of equations.

3.1.1 Current-Voltage Characteristics

Almost all of the analog circuits designed in this work operate in subthreshold, where the gate-source voltage v_{gs} is well below the MOSFET threshold voltage V_{th0}. The long channel model of the subthreshold drain current can be written as

$$i'_{ds} = i_{ds0}\text{exp}\left(\frac{\zeta_1 v_{gs} - V_{th0}}{nV_T}\right), \tag{3.1}$$

where

$$i_{ds0} = \zeta_2 \beta (n-1) V_T^2, \tag{3.2}$$

and $\beta \equiv \mu_0 C_{ox} W/L$ is the current factor, μ_0 is the low field carrier mobility, C_{ox} is the oxide capacitance, W is the channel width, L is the channel length, n is the subthreshold slope factor, V_T is the thermal voltage, v_{gs} is the gate-source voltage, ζ_1 and ζ_2 are fitting parameters, and V_{th0} is the threshold voltage. It is assumed that the drain-source voltage $v_{ds} \geq 4V_T \approx$1 mV at room temperature (i.e. the device is saturated). Note that this model is most accurate when $V_L \leq v_{gs} \leq V_M$, where V_L and V_M define the depletion/weak inversion and weak inversion/moderate inversion borders, respectively. This work uses $V_L =$ 0.2 V and $V_M = V_{th0} -$ 0.1 V. Although these values can be found rigorously based on the flat band and Fermi voltages, the chosen values of V_L and V_M for the model being developed. Now, (3.1) must be modified to match the results of SPICE simulations. First, the log (base

10) is taken of both sides, and then two additional fitting parameters ζ_3, ζ_4 are introduced:

$$\log(i_{ds}) = \zeta_1 \log(e) \frac{v_{gs}}{nV_T} + \log\left(\beta(n-1)V_T^2\right) - \log(e)\frac{V_{th0}}{nV_T} + f(\zeta_2, \zeta_3, \zeta_4). \tag{3.3}$$

Now, the ζ parameters, as well as the form of f, are adjusted to match the slope and intercept of the log of the simulated voltage transfer characteristics in weak inversion. The idea is illustrated in Figure 3.1 for an NMOS device with, $W = L = 45$ nm, $v_{ds} = 1.1$ V, and $V_{sb} = 0$ V. Note that, unless stated otherwise, the source-body voltage $V_{sb} = 0$ V for all devices in this work.

Parameter ζ_1 can be found easily since the slope in the weak inversion region has very little dependence on any of the free device parameters (W, L, v_{ds}, v_{gs})[1]. A straight line is fit to $\log(i_{ds})$ in the domain $V_L \leq v_{gs} \leq V_M$ for several different sets of $\{W, L, v_{ds}\}$, resulting in multiple values of $\zeta_1 = SnV_T/\log(e)$, where S is the slope of the line. Then the final value of ζ_1 is taken as the median over all of the parameter sets. In particular, W and L are varied from 45 nm to 4500 nm, while v_{ds} is varied from 0.1 V to 1.1 V. The results are shown in Table 3.1. Notice that there is little variation except in the case where $L = 4500$ nm and $v_{ds} = 1.1$ V. It is important to point out that the circuits presented in this work will never have such extreme aspect ratios, and they are only included for completeness.

The function f is a little more complicated because the intercept depends on L and v_{ds}:

$$f(\zeta_2, \zeta_3, \zeta_4) = \log(\zeta_2) + \frac{\log(e)\zeta_3 \exp(-\zeta_4 L)\, v_{ds}}{nV_T}. \tag{3.4}$$

[1]There is some dependence, however. For example, gate leakage will be related to the area of the device.

Table 3.1: ζ_1 parameters for different values of W, L, and v_{ds}.

			ζ_1	
W [nm]	L [nm]	$\lvert v_{ds}\rvert$ [V]	NMOS	PMOS
45	45	0.1	0.97	0.97
45	45	1.1	0.92	0.92
45	4500	0.1	0.94	0.92
45	4500	1.1	0.82	0.72
4500	45	0.1	0.97	0.97
500	45	1.1	0.92	0.92
4500	4500	0.1	0.97	0.98
4500	4500	1.1	0.97	0.97
		Median:	0.96	0.95

These ζ parameters were found by simulating the devices at a fixed $|v_{gs}| = 0.4$ V (approximately midway between V_L and V_M) while sweeping L and v_{ds}. Then, surfaces of the form (3.3) were fit to the data. The results are shown in Figure 3.2, and the corresponding ζ parameters are listed in Table 3.2, along with all of the other parameters required for the simplified model. Note that (3.3) can also be written as

$$i_{ds} = i'_{ds}\text{exp}\left(\frac{\zeta_3 \text{exp}\left(-\zeta_4 L\right) v_{ds}}{nV_T}\right) = i'_{ds}\text{exp}\left(\Theta\left(L\right) v_{ds}\right) = i'_{ds}\Lambda\left(v_{ds}, L\right). \tag{3.5}$$

In this form, it is easier to see the effect of v_{ds} and L on the I-V characteristics. For example, when the channel length is small, there is an exponential dependence on v_{ds}, as is the case in drain-induced barrier lowering. However, when the channel length is larger, the v_{ds} dependence becomes weak, closer to the effect of channel length modulation.

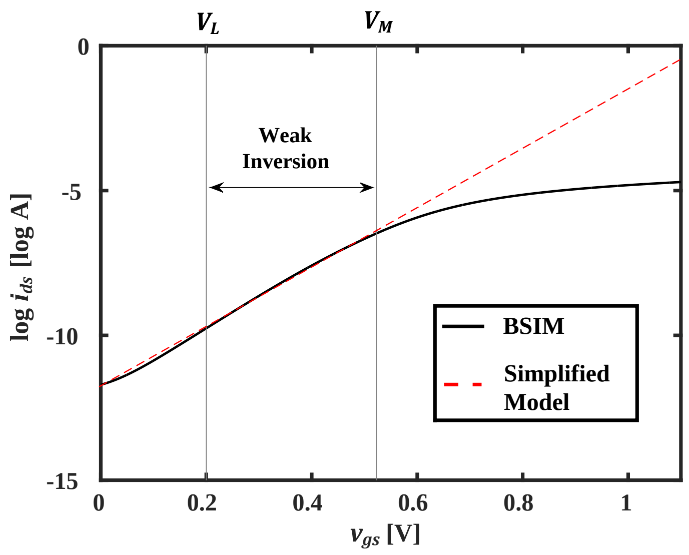


Figure 3.1: Logarithm of the NMOS drain current versus v_{gs} ($W = L = 45$ nm, $v_{ds} = 1.1$ V, $V_{sb} = 0$ V). The solid curve shows the result from SPICE simulation using the BSIM model, while the dashed curve shows the proposed semi-empirical weak inversion model.

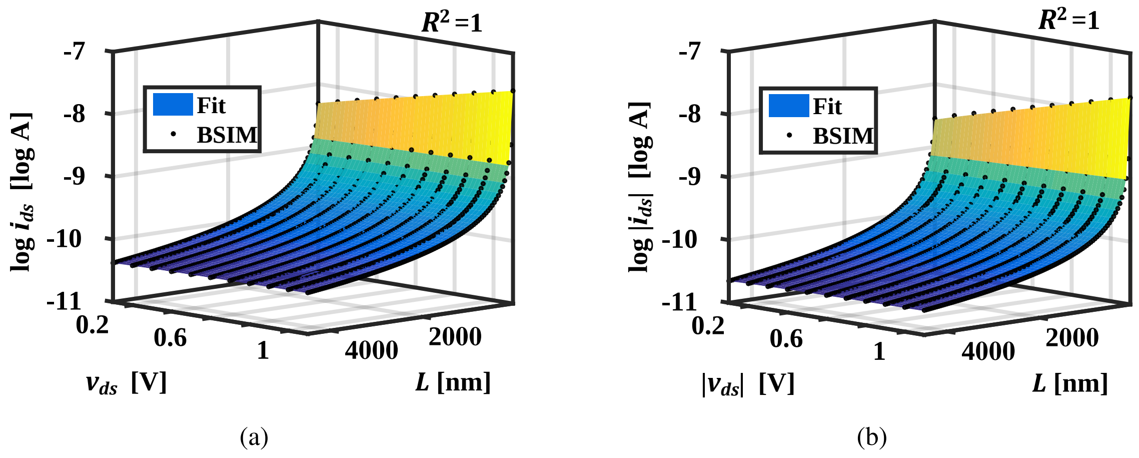


Figure 3.2: Fitting the L and v_{ds}-dependent ζ parameters for (a) NMOS and (b) PMOS devices.

3.1.2 Output Impedance

A critical characteristic of a MOSFET is its output impedance r_o, which is defined as

$$r_o \equiv \left(\frac{\partial i_{ds}}{\partial v_{ds}} \right)^{-1}. \tag{3.6}$$

Table 3.2: Low power PTM 45 nm MOSFET parameters.

Parameter	**NMOS**	**PMOS**	**Description**
C_{ox} [F/m^2]	0.019	0.019	Oxide capacitance
V_{th0} [V]	0.62	-0.59	Threshold voltage at $V_{sb} = 0$
μ_0 [m^2/Vs]	0.049	0.021	Low field mobility
n	1.5	1.5	$n \equiv \left(\frac{\mathrm{d}\psi_{sa}}{\mathrm{d}V_{gb}}\right)^{-1}$
ζ_1	0.96	0.95	Fitting parameter
ζ_2	6.742	3.838	Fitting parameter
ζ_3	0.2756	0.3425	Fitting parameter
ζ_4	3.3956×10^7	3.4489×10^7	Fitting parameter
$A_{V_{th0}}$ [mVμm]	1.8	1.8	Process constant describing threshold voltage mismatch

From (3.5), r_o can be written as

$$r_o = \frac{1}{i_{ds}\Theta\left(L\right)}. \tag{3.7}$$

The drain current and channel length have opposite effects on the output impedance, as expected.

3.1.3 Diode-Connected MOSFETs

This work makes frequent use of diode-connected MOSFETs, where the device's gate is connected to its drain. Equivalently, $v_{gs} = v_{ds}$, so (3.5) can be solved in terms of i_{ds} as

$$v_{gs} = \frac{nV_T \ln\left(\frac{i_{ds}}{i_{ds0}}\right) + V_{th0}}{\zeta_1 + nV_T\Theta\left(L\right)}, \tag{3.8}$$

where $\ln(\cdot)$ is the natural logarithm.

3.1.4 ON Resistance

An important superthreshold characteristic of the MOSFETs used in this work is their ON resistance, which is measured when $|v_{gs}|$ and $|v_{ds}|$ are at their maximum values (i.e. $V_{DD} + V_{SS}$). Figure 3.3 shows the ON resistance of NMOS and PMOS devices with varying channel widths. All channel lengths are equal to $L = 45$ nm. The large values of the ON resistance result in large IR drops when MOSFETS are used as analog switches (e.g. transmission gates or pass transistors). This has a direct implication for the maximum conductance G_{mon} that a memristor can have in order to program it using a transistor circuit.

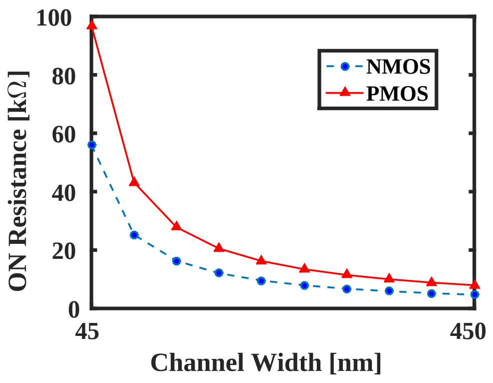


Figure 3.3: ON resistance of 45 nm low-power PTM MOSFETs.

3.1.5 Process Variations

The accuracy (target output voltage or current vs. actual output voltage or current) of CMOS analog circuits is highly sensitive to physical parameters that vary across devices on the same die. In general, mismatch in both β and V_{th0} parameters are modeled in order to capture these effects. However, in subthreshold operation, variations in V_{th0} have a dominant exponential effect on the variations in circuit behavior, so variations in β can be ignored. From Pelgrom's work, it can be shown that the variance of the difference in threshold voltage V_{th0} between two closely-spaced MOSFETs is given as [62, 63]

$$\sigma^2(\Delta V_{th0}) = \sigma^2(\delta V_{th01}) + \sigma^2(\delta V_{th02}) = \frac{A^2_{V_{th0}}}{2W_1L_1} + \frac{A^2_{V_{th0}}}{2W_2L_2}, \tag{3.9}$$

where $A_{V_{th0}}$ is a process-dependent constant. In this work, a value of $A_{V_{th0}} = 1.8$ mVμm was used for both NMOS and PMOS devices, which follows from the empirical law $A_{vth0} =$ (1 mVμm/nm)t_{ox} [64]. This value of $A_{V_{th0}}$ is also in close agreement with data published from Intel's 45 nm process [65]. Equation (3.9) reveals that transistors with larger areas will match better, so there is an inherent area/reliability tradeoff. This model of the threshold voltage variation will be used to assess the random variations in NMS primitive circuits.

3.2 Memristor Models

NMS design choices are guided by the characteristics of a target memristor technology. Therefore, a memristor model must be chosen that accurately reflects the behavior of fabricated devices. The best choice would be a model that is in use by a majority of other NMS researchers. This would allow for a fair comparison of state-of-the-art NMS designs to those proposed in this work. However, each group bases their designs on different models, making it impossible to identify one standard choice. Even more difficult is the fact that memristor materials, fabrication processes, and theory have been rapidly evolving.

To overcome these challenges, a semi-empirical model is developed that is strongly rooted in memristor device physics, but flexible enough to account for static and dynamic behavior which are not yet fully understood. Highlighted here are the two key theoretical components that must be modeled. The first is the I-V relationship when the memristor is in a fixed state. The second is the evolution of the memristor state with the application of an electric field.

This work makes use of two types of memristors with different I-V and switching characteristics. The first is a silver chalcogenide device that exhibits incremental conductance switching and a weak exponential I-V relationship. The second is a bi-stable (two conductance states) CBRAM device that has stochastic switching behavior. Critically, both devices are compatible with CMOS fabrication and can be integrated into a back end of line (BEOL) process [53, 66]. This enables the heterogeneous design of MOSFET/memristor circuits.

3.2.1 Silver Chalcogenide Memristor

A memristor is a thin, nominally insulating film, sandwiched between two electrodes. How insulating the film is depends on its state, which effectively modifies the film's band gap in different regions. Figure 3.4 shows a memristor with a particular distribution of defects in the insulating film. A simplified energy band diagram of a 1-dimensional slice (dashed rectangle) is shown on the bottom. It is assumed that a voltage v_m has been applied across the electrodes, from the right to the left. If the insulator film were actually a conductor, then electrons would drift in the electric field from the left electrode to the right electrode. However, the insulator imposes a potential energy barrier which has to be overcome. Notice that there are valleys in the energy barrier corresponding to local defects. If the distance a between the valleys is small enough, then electrons can tunnel between defect sites, giving

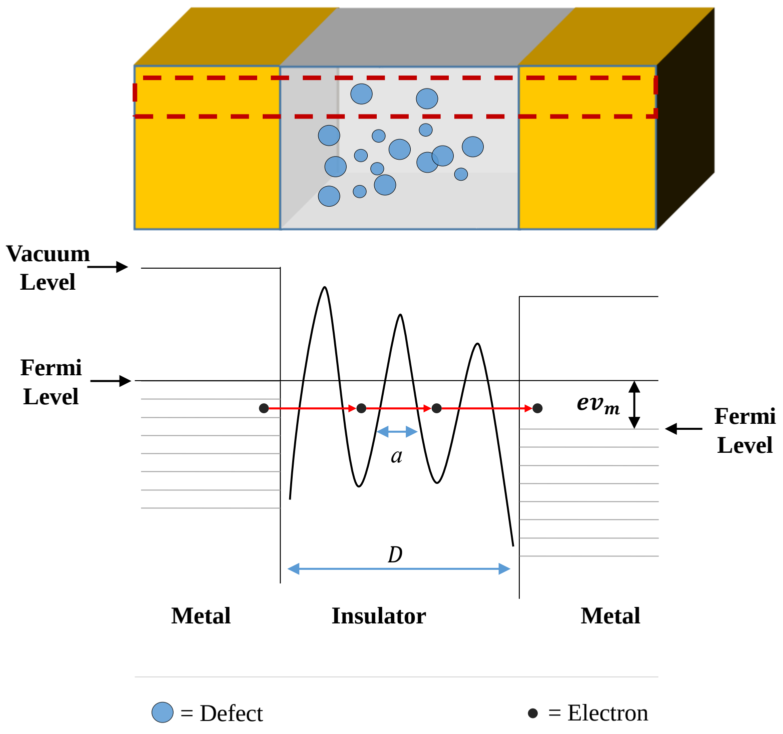


Figure 3.4: Memristor with energy band diagram showing dips in the energy level of the insulator corresponding to defect states.

rise to a current. One of the first to accurately model this phenomenon was Simmons in 1963 [67]. Outlined here are the key points of his derivation and how it leads to a semi-empirical memristor model that shows excellent agreement with a wide variety of experimental devices.

To start, consider the time-independent Schrödinger equation in one dimension:

$$-\frac{\hbar^2}{2m}\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} + U(x)\psi = E\psi, \tag{3.10}$$

where $\hbar$ is Planck's constant divided by 2π, m is the mass of an electron, ψ is the wave function, x is the horizontal dimension, U is the potential energy, and E is the total energy. Now, consider an electron in Figure 3.4. On approaching one of the potential hills within the insulator, part of its incident wave function will be reflected, and part of it will be transmitted to the next valley. Therefore, the probability that the electron will tunnel between two valleys will be the ratio of the squares of the wave function amplitudes of the incident wave and the transmitted wave. Using the WKB approximation, it can be shown that the tunneling probability is [68]

$$T\left(E_x\right) \approx \exp\left(-\frac{2}{\hbar}\int\limits_{0}^{a} |p(x)|\,\mathrm{d}x\right), \tag{3.11}$$

where p is the electron momentum, and $x = 0$ is taken to be the left side of the energy barrier in question. Now, the total probability of tunneling will be the sum of all the probabilities at each energy level E times the number of electrons in that energy level, which can be described by Fermi-Dirac statistics. Simmons showed that the total probability can be written as

$$T = \exp\left(\alpha_2 a\sqrt{\bar{E}_a}\right), \tag{3.12}$$

where α_2 is a constant and $\bar{E}_a$ is the mean height of the barrier, or activation energy. Now, the electron current can be written as

$$i_m \propto \nu\left(T_R - T_L\right), \tag{3.13}$$

where ν is the attempt-to-escape frequency, T_R and T_L are the probabilities of an electron tunneling to the right and left, respectively. Of course, the difference between $\bar{E}_{aR}$ and $\bar{E}_{aL}$ will depend on the electric field, which creates higher barriers at the cathode (left electrode), and lower barriers at the anode (right electrode). With a few more approximations, it can be shown that [67, 69]

$$i_m = \alpha_1 \mathrm{exp}\left(\alpha_2\left(\bar{E}_a - \sqrt{\frac{F_0 a}{2}}\right)\right)\sinh\left(\sqrt{\alpha_3 v_m a^2}\bar{E}_a - F_0 a/2\right), \tag{3.14}$$

where α_1, α_2, and α_3 are constants, and F_0 is the electric field at $x = 0$.

The state of a memristor γ can be defined in several ways. In this work, γ characterizes the distribution of defects within the memristor film. Like electrons and holes, defects, whether they are interstitial ions (Frenkel defects), vacancies (Schottky defects), etc., can drift in an electric field. Suppose, for example, that two interstitial sites are separated by a potential energy barrier of width a and height E_a. The probability that an interstitial ion will move from one site to the other is related to the fraction of time that its energy E is above E_a, which is given by Boltzmann's law as $\exp\left(-\frac{E_a}{kT}\right)$. Now, when an electric field is applied to the film, it is easy to show that the mean drift velocity and, hence, the rate of change of the memristor state can be expressed as [70]

$$\frac{\mathrm{d}\gamma}{\mathrm{d}t} = \chi\left(v_m(t)\right) = \nu a \mathrm{exp}\left(-\frac{E_a}{kT}\right)\sinh\left(\frac{qav_m/D}{kT}\right), \tag{3.15}$$

where ν is the lattice vibration frequency. Notice that at low voltages and nominal temperatures (e.g. room temperature), changes in γ are very small, resulting in non-volatile behavior.

Equations (3.14) and (3.15) can be modified to create a semi-empirical memristor model, with fitting parameters that can be adjusted to fit a wide range of experimental devices. This idea has been previously explored by Yakopcic et al., [71]. In this work, a few small adjustments are made in order to make the model more amenable to constrained non-linear curve fitting. First, the state variable γ must be incorporated into the (3.14). Since γ characterizes the distribution of defects within the memristor film, it should have some dependence on the tunneling barrier width a, which appears in the exponential and the sinh functions in (3.14). However, only including the exponential dependence still yields an accurate model with fewer fitting parameters. This leads to the following expression for the memristor I-V characteristic:

$$i_m = \begin{cases} \gamma G_{mon} v_m + (1-\gamma) G_{moff} \xi_1^+ \sinh\left(\frac{v_m}{\xi_1^+}\right), v_m \geq 0 \\ \gamma G_{mon} v_m + (1-\gamma) G_{moff} \xi_1^- \sinh\left(\frac{v_m}{\xi_1^-}\right), v_m < 0 \end{cases}, \tag{3.16}$$

$\xi_1^{+(-)}$ are fitting parameters for the positive (negative) portion of the I-V characteristic. Notice that, when v_m is small, $\sinh\left(v_m/\xi_1^{+(-)}\right) \approx v_m/\xi_1^{+(-)}$, so

$$i_m \approx \left[\gamma G_{mon} + (1-\gamma) G_{moff}\right] v_m, \quad v_m \approx 0, \tag{3.17}$$

which agrees with other models, such as the linear ionic drift model [19]. In contrast, the model presented in [27] has $i_m \propto \gamma \sinh(v_m)$, implying that the state variable must be non-zero to have any current flow, which is not in agreement with other models.

Next, (3.15) is modified to generate a tractable expression for the evolution of γ. First, the activation energy E_a is taken as a hard threshold voltage, below which there will be no change in state. The change in the state variable becomes

$$\frac{\Delta\gamma}{\Delta t} = \chi\left(v_m(t)\right) = \begin{cases} \xi_4^{+}\sinh\left(\xi_5^{+}v_m(t) - \xi_6^{+}V_{tp}\right) f_{win}\left(\gamma\right), & v_m > V_{tp} \\ \xi_4^{-}\sinh\left(\xi_5^{-}v_m(t) - \xi_6^{-}V_{tn}\right) f_{win}\left(\gamma\right), & v_m < V_{tn} \\ 0, & \text{otherwise} \end{cases}, \tag{3.18}$$

where V_{tp} and V_{tn} are the positive and negative memristor threshold voltages, which are usually easy to identify from experimental data. In addition, the $\xi_i^{+(-)}$ parameters are for fitting, and f_{win} is a window function that accounts for degradation in state variable evolution near the boundaries of the film. The use of window functions in memristor models was first proposed by Biolek et al. [23], where f_{win} was a concave geometric function. This work adopts an exponential decay for the window function, as in [72]. The function takes the form

$$f_{win}\left(\gamma\right) = \begin{cases} \exp\left[-\xi_2^{+}\left(\gamma - \xi_3^{+}\right]\right)\frac{1-\gamma}{1-\xi_3^{+}}, & v_m \geq 0, \gamma \geq \xi_3^{+} \\ \exp\left[\xi_2^{-}\left(\gamma - \xi_3^{-}\right)\right]\frac{\gamma}{\xi_3^{-}}, & v_m < 0, \gamma \leq \xi_3^{-} \\ 1, & \text{otherwise} \end{cases}, \tag{3.19}$$

where $\xi_i^{+(-)}$ are fitting parameters. For $v_m \geq 0$, $f_{win} = 1$ from $\gamma = 0$ up to $\gamma = \xi_3^{+}$, then exponentially decays, and becomes 0 at $\gamma = 1$. For $v_m \leq 0$, $f_{win} = 1$ from $\gamma = 1$ down to $\gamma = \xi_3^{-}$, then exponentially decays, and becomes 0 at $\gamma = 0$. These boundary conditions ensure that γ stays between 0 and 1. Physically, this models the fact that defects generally won't drift into the electrode materials.

The model described above was fit to experimental data from a silver chalcogenide device published in [66]. The material stack is shown in Figure 3.5. Figure 3.6 shows the

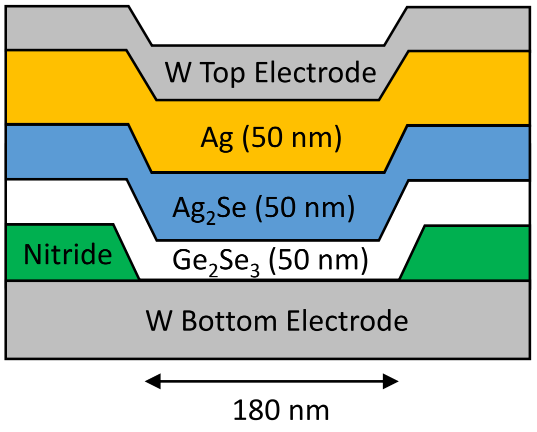


Figure 3.5: Silver chalcogenide memristor stack (adapted from [66]).

memristor current versus time when a sinusoidal voltage is applied [73]. The top plot shows the voltage across the memristor, which was placed in series with a 1.6 kΩ resistor. The bottom plot shows the memristor current vs. time for 8 different devices. Inter-device and inter-cycle variations are observed. The devices show small inter-cycle variations, which are not modeled in this work. Instead, the mean is taken over the five cycles to generate I-V characteristics for each of the 8 devices. Since the inter-cycle variations are small, the averaging does not cause significant distortion to the characteristic features.

A single I-V curve needs to be generated from the 8 devices to determine the parameters for the nominal memristor model. One approach is to take the mean response over all of the devices. However, given the large inter-device variation, especially in G_{mon} and G_{moff}, this leads to smoothing that does not faithfully represent the behavior of any one device. Instead, the device with values of G_{mon} and G_{moff} that are closest to the mean is chosen as the nominal device. Figure 3.7 shows the *on* and *off* conductances for each of the 8 devices. Horizontal dashed lines indicate the mean values, $\langle G_{mon}\rangle$ and $\langle G_{moff}\rangle$. Above each device is the sum of the differences (in percentage) of that device's *on* and *off* conductances from

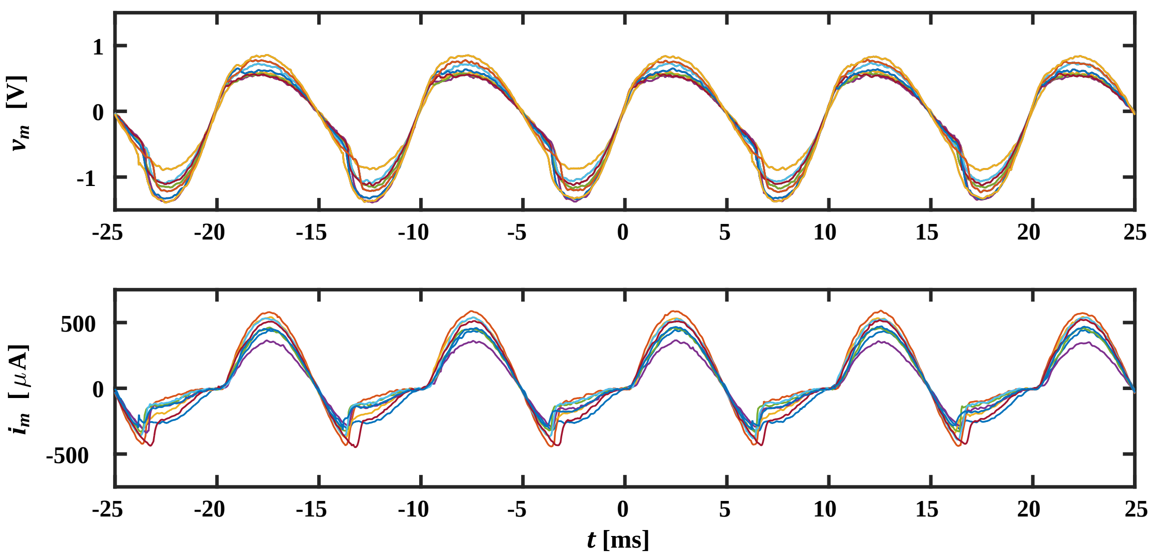


Figure 3.6: Silver chalcogenide current versus time resulting from the application of a sinusoidal voltage. (Top) Voltage across the memristor vs. time. (Bottom) Current flowing through the memristor vs. time for 8 devices [73].

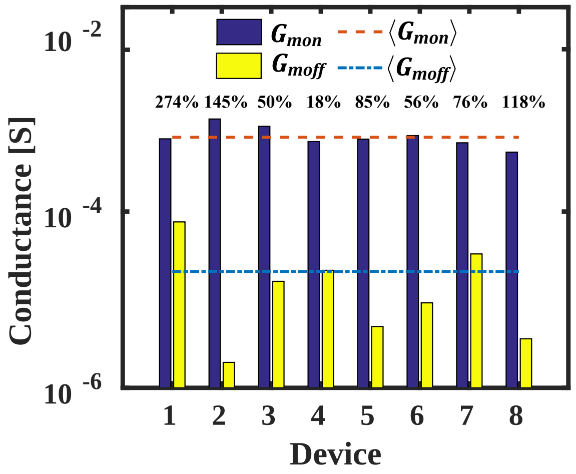


Figure 3.7: *on* and *off* conductances of 8 silver chalcogenide memristor devices. Dashed lines show the mean values. Shown above the bars for each device is the sum of the percent difference between that device's *on* and *off* conductance and the mean values.

the mean values:

$$\%\text{diff} \equiv 100\% \left(\frac{|G_{mon_i} - \langle G_{mon} \rangle|}{\langle G_{mon} \rangle} + \frac{|G_{moff_i} - \langle G_{moff} \rangle|}{\langle G_{moff} \rangle} \right), \tag{3.20}$$

where i is the device number. Device 4 has the smallest sum of differences from the means, so it is used as the nominal device. Table 3.3 shows the fitting parameters, which were found in MATLAB using a constrained non-linear optimization technique. Figure 3.8 shows the I-V plot of the experimental data and the model. The model matches the experimental data with a mean error of 25%.

Table 3.3: Model parameters for a silver chalcogenide memristor.

Parameter	**Value**	**Bounds**
ξ_1^+	0.9934	$[0,+\infty]$
ξ_2^+	2.5275	$[0,+\infty]$
ξ_3^+	0.3394	[0,1]
ξ_4^+	113.5	$[0,+\infty]$
ξ_5^+	3.8153	$[0,+\infty]$
ξ_6^+	-2.0429	$[-\infty,+\infty]$
ξ_1^-	0.2727	$[0,+\infty]$
ξ_2^-	4.2894	$[0,+\infty]$
ξ_3^-	0.4837	[0,1]
ξ_4^-	106.2875	$[0,+\infty]$
ξ_5^-	4.0992	$[0,+\infty]$
ξ_6^-	-3.0634	$[-\infty,+\infty]$
V_{tp}	0.4 V	
V_{tn}	-0.55 V	
G_{mon}	$(1800\Omega)^{-1}$	
G_{moff}	$(4.637 \times 10^4\Omega)^{-1}$	
g	25.76	

The evolution of the memristor state variable γ is critical to an NMS's learning process. Figure 3.9 shows several different state transitions when multiple write pulses of magnitude v_w and duration t_w are applied to the silver chalcogenide device. In Figure 3.9(a), $|v_w|$ =0.75 V, and t_w=1 ns. The transitions from 0 to 1 and 1 to 0 both take $\approx 10^7$ pulses. When the pulse magnitude is increased to 1 V (Figure 3.9(b)), the number of pulses required to transition is reduced by 2-3 orders of magnitude. Figures 3.9(c) and 3.9(d) show

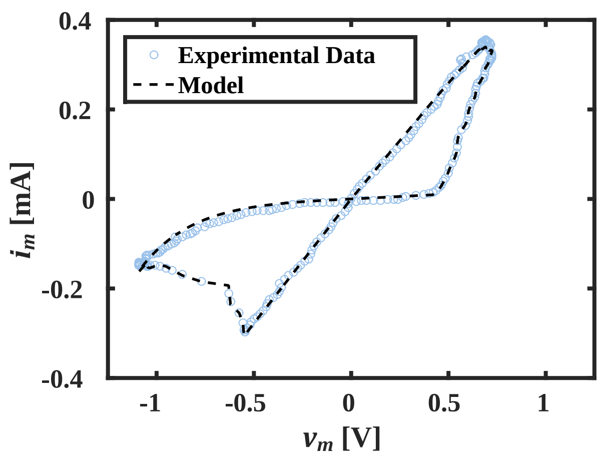


Figure 3.8: Silver chacogenide memristor I-V characteristic: Experimental data [73] and semi-empirical model.

the evolution of γ when the pulse duration is increased to 1 μs, with pulse magnitudes of 0.75 V and 1 V, respectively. The number of write pulses required to change γ between the two extreme values is significantly reduced. However, the energy consumption per write pulse will obviously increase with increased t_w and v_w. Therefore, there will be a tradeoff between an NMS's power consumption and the rate at which it can adapt/learn. This is explored in more detail throughout this dissertation. Finally, it should be noted that there is an inherent asymmetry in the memristor switching characteristics, which is evident from all of the plots in Figure 3.9. In fact, most memristive devices show some form of switching asymmetry. As a consequence, synapses and other memristor circuits in an NMS will generally adapt at different rates depending on whether they or being potentiated or depressed.

3.2.2 CBRAM Memristor

CBRAM memristors operate on the principle of reversible electrochemical reactions. The device is an MIM type two-terminal structure, with an active solid electrolyte sandwiched

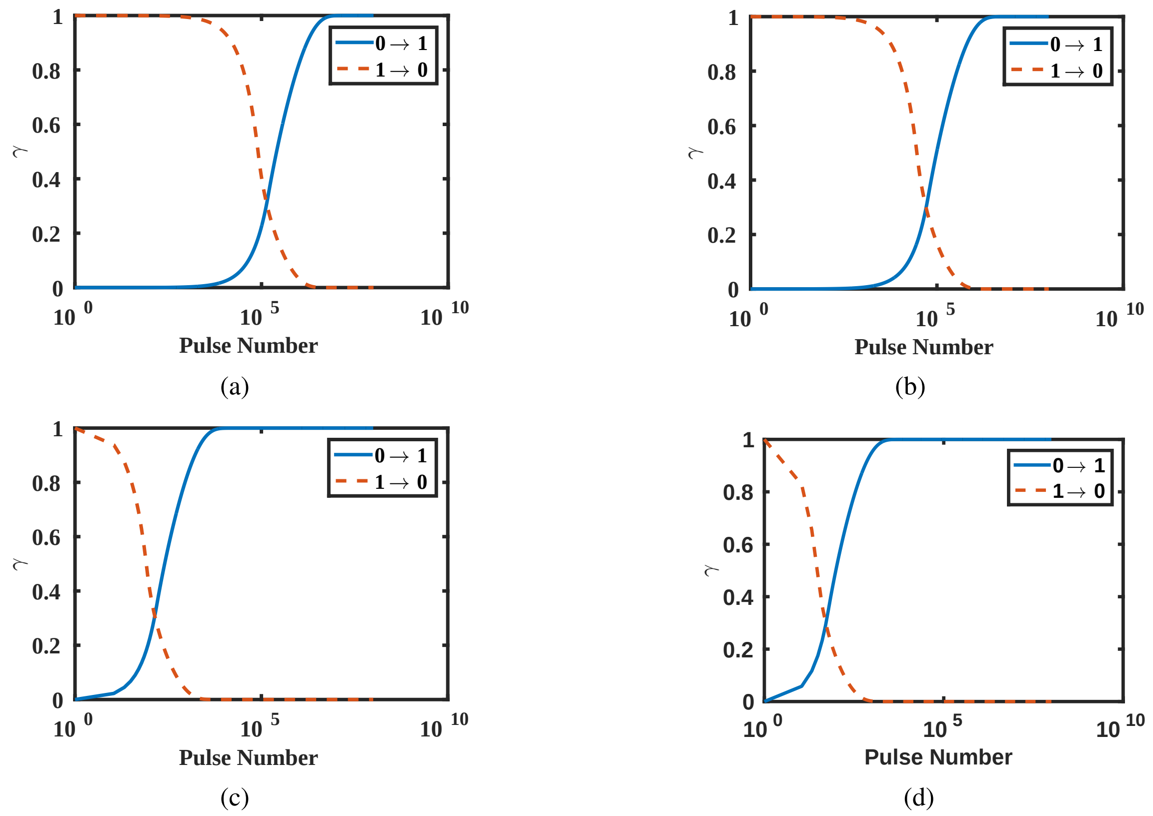


Figure 3.9: Change in state variable γ vs. number of applied write pulses. (a) $|v_w| = 0.75$ V, $t_w = 1$ ns. (b) $|v_w| = 1.0$ V, $t_w = 1$ ns. (c) $|v_w| = 0.75$ V, $t_w = 1\mu$s. (d) $|v_w| = 1.0$ V, $t_w = 1\mu$s.

between two metal electrodes [74]. In particular, the devices studied in this work [2] consisted of an active Ag top electrode (anode), inert W bottom electrode, and 30 nm thick GeS_2 electrolyte [75] (See Figure 3.10). On application of a positive write voltage v_w, Ag atoms from the anode are oxidized and the resulting ions enter the switching layer where they drift to the cathode. At the cathode, the Ag ions are reduced. Over time there is formation of a conductive Ag filament in the GeS_2 layer, changing the CBRAM conductance G_m to a high conductance *on* state. On reversing the polarity of the applied voltage, the conductive Ag filament gets dissolved, changing G_m to a low conductance *off* state [76].

[2]The CBRAM devices used for this work were fabricated and tested at CEA-LETI-Grenoble.

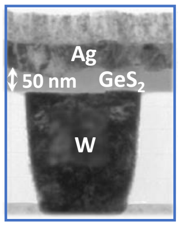


Figure 3.10: SEM image of the CBRAM device used in this work [75].

Previously, CBRAM devices have been modeled by considering the growth/dissolution rate of the metallic filament in the switching layer [77, 78]:

$$\frac{\mathrm{d}h}{\mathrm{d}t} = v_h \exp\left(\frac{-E_a}{kT}\right) \sinh\left(\frac{zqEa}{2kT}\right) \tag{3.21}$$

$$\frac{\mathrm{d}r}{\mathrm{d}t} = v_r \exp\left(\frac{-E_a}{kT}\right) \sinh\left(\frac{\beta q v_w}{kT}\right). \tag{3.22}$$

In (3.21), h is the filament height or length, E_a is the activation energy required for the metal ions to drift, k is Boltzmann's constant, T is temperature, q is the ion charge, Z is the number of charged ions, E is the electric field, and v_h and a are fitting parameters. In (3.22), r is the radius of the metallic filament, V is the applied voltage, and v_r and β are fitting parameters. Finally, the *on* (G_{mon}) and *off* (G_{moff}) conductance values can easily be obtained from the geometries of the conductive filament and CBRAM cell, as well as the resistivities of the materials in the CBRAM stack. All of the model parameters for Ag/GeS$_2$/W devices can be found in [77].

Recently, Suri et al., showed that CBRAM devices switch stochastically under weak programming conditions (small applied write voltages and programming durations) [53]. For example, it was shown that write voltages around 1.5 V applied for 500 ns can be used to achieve switching probabilities p_{switch} in Ag/GeS$_2$/W devices that are below 0.1.

Later, it will be shown that this probabilistic switching phenomena is very useful for designing and training CBRAM-based synapse circuits. Unfortunately, this behavior is not easily described within the theoretical framework discussed above. Therefore, a simple phenomenological model is developed based on the experimental data presented in [53]. For simplicity, it is assumed that the CBRAM switching behavior is symmetric. That is, the metallic filament growth and dissolution rates are equal when voltages of equal magnitude and opposite sign are applied to the CBRAM device. This assumption is supported by (3.21) and (3.22) since the $\sinh$ function is symmetric about the origin. Figure 3.11 shows the switching probability of CBRAM devices versus the applied flux ($\phi = \int v_w \mathrm{d}t$) from the data presented in [53]. The data are fit to a log-normal cumulative distribution function (CDF):

$$p_{switch} = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left[\frac{\ln\phi - \mu}{\sqrt{2}\sigma}\right] \tag{3.23}$$

erf is the error function, and μ and σ are fitting parameters which are associated with the corresponding normal distribution. Other functions such as the gamma and beta CDFs also fit the data well. However, it was shown in [53] that the CBRAM *on* and *off* conductance states are log-normally distributed, so it is hypothesized that the processes governing the switching probability are also associated with a log-normal distribution. It is important to note that the p_{switch} model presented here is only valid if the applied write voltage is large enough so that the Ag ions can drift inside the GeS_2 lattice. It is estimated in [78] that the required activation energy is 0.4 eV. This work uses write voltages that have experimentally been shown to induce change in the device's conductance states, thereby ensuring that the model is valid.

The overall CBRAM model is described by

$$\Delta\gamma = S\text{sgn}\phi\left(\gamma - \text{sgn}\phi\right) \tag{3.24}$$

and

$$G_m := \begin{cases} G_m & : \text{sgn}\left(\Delta\gamma\right) = 0 \\ G_{moff} & : \text{sgn}\left(\Delta\gamma\right) = -1 \\ G_{mon} & : \text{sgn}\left(\Delta\gamma\right) = +1 \end{cases} . \tag{3.25}$$

Here, S is a Bernoulli-distributed random variable with a success probability equal to p_{switch}. The sgn function is -1, 0, or 1 for negative, zero, or positive arguments, respectively. The variable $\gamma \in \{-1, 1\}$ represents the state of the CBRAM device, with -1 corresponding to an *off* state and 1 corresponding to an *on* state. Note that if the sign of the applied flux is the same as the sign of the state variable, then the device will not switch. This is a simplified model as, for example, applying a positive flux to a device already in the *on* state may still create some small changes in the conductance. From (3.25), it can be seen that if the state variable changes to a negative (positive) value, then the device's conductance will change to G_{moff} (G_{mon}) which is sampled from a log-normal distribution. The variations in these conductances and the switching probabilities are discussed below.

3.2.3 Process Variations

There are several process variations and wearout mechanisms that affect memristor I-V and switching characteristics. This work considers two variations that are particularly important when using a memristor as a synapse in an NMS. The first is the variation in the

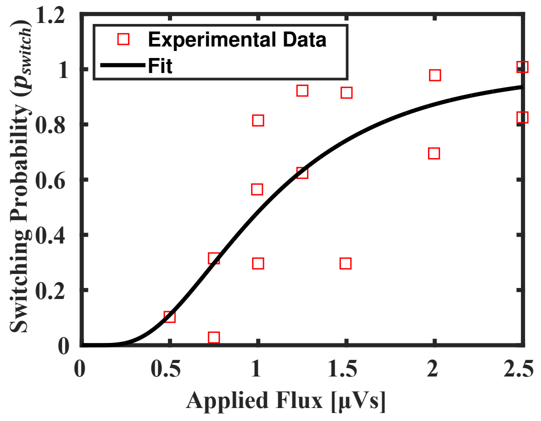


Figure 3.11: CBRAM switching probability vs. applied flux.

minimum and maximum memristor conductances, G_{moff} and G_{mon}. These variations are caused by various non-idealities in the fabrication process such as line edge roughness, memristor thickness fluctuation, and random discrete doping, among others [79–82]. In general, the most important characteristic to control during fabrication is the switching layer's defect profile [13, 15] (e.g. number of vacancies, interstitial defects, grain boundaries, etc.), which affects the device's *on* and *off* conductances as well as its switching time, and threshold voltages. The effects of these variations on G_{mon} and G_{moff} have been estimated in previous work. For example, in [81], the authors present a device with nominal conductance values of $G_{mon} = 1.0\times10^{-2}$ ℧ and $G_{moff} = 6.25\times10^{-5}$ ℧ and estimate the 3σ variation in the *on* and *off* conductances to be $\approx 7\%$. In this work, the variations in G_{mon} and G_{moff} were measured from the Ag chalcogenide and CBRAM memristor experimental data. The results are shown in Table 3.4. The variations in these maximum and minimum conductances will directly affect the maximum and minimum weight values that can be achieved in synapse circuits, which are discussed in the next section. This can be particularly problematic when the network weights are limited to a small range, such as

Table 3.4: Memristor variation parameters for Ag chalcogenide and CBRAM devices.

Device Parameter	**Ag Chalcogenide**	**CBRAM**
$\mu\,(p_{switch})$ [ln $(\mu$Vs)]	N/A	0.024
$\sigma\,(p_{switch})$ [ln $(\mu$Vs)]	N/A	0.587
$\mu\,(G_{moff})$ [S]	2.08×10^{-5}	1.12×10^{-6}
$\sigma\,(G_{moff})$ [%]	119	128
$\mu\,(G_{mon})$ [S]	7.26×10^{-4}	0.38×10^{-3}
$\sigma\,(G_{mon})$ [%]	28.3	9.46
$\sigma\,(t_w)$ [%]	10	N/A

[-1,1], for the following reason: The distribution of weights within a trained network with unrestricted weight values is typically Gaussian [83]. However, when the weights are restricted, the distribution changes such that most of the values lie at the extrema of the range. Therefore, many memristive synapses affected by process variations may not be able to be programmed to an ideal weight value for a specific NMS application.

In addition to G_{mon} and G_{moff}, the write time t_w of a memristor (the time that it takes to change its conductance between two values) is also affected by process variations [81]. In particular, variations in the thickness of the memristor film will have a non-linear effect on the write time. In [81], the authors estimate variations in the film thickness to be $\approx 2\%$. Assuming that the write time is proportional to the inverse of the thickness squared, and the nominal thickness is ≈50 nm, the standard deviation of the write time will be 4%. However, a more conservative estimate of 10% is used in this work to account for other factors that will affect the write time, including variations in defect mobility, etc. The variations in write time will affect the learning/programming rate of the weights within the NMS. For CBRAM devices, there are variations in the switching probability p_{switch} instead of t_w. The variation parameters for t_w and p_{switch} are shown in Table 3.4.

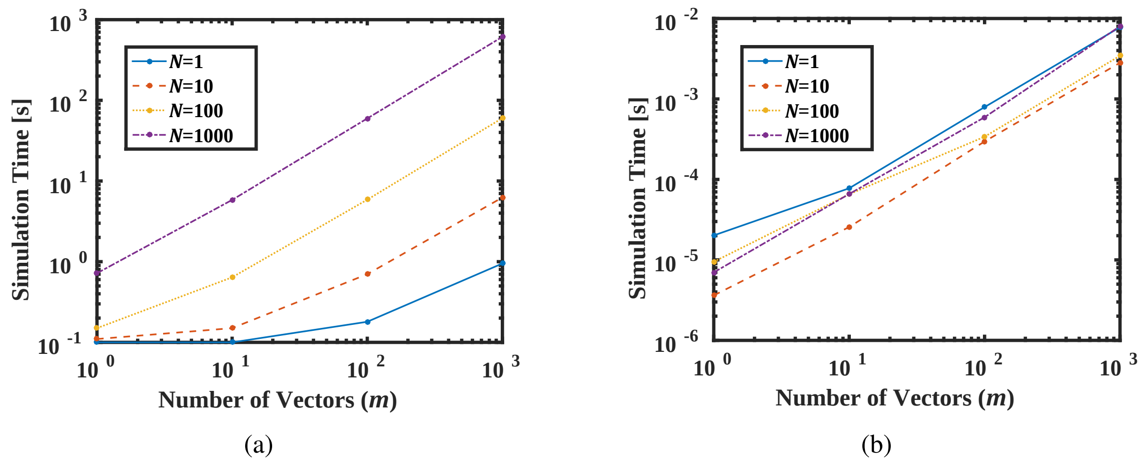


Figure 3.12: Simulation times for a single-layer perceptron with different numbers of inputs N and test vectors m applied. (a) Mean HSPICE simulation time. (b) Mean MATLAB simulation time.

3.3 Simulation Strategy

The most accurate way to verify an NMS is to simulate it entirely in SPICE. SPICE simulators use nodal analysis to to numerically solve circuits composed of linear and non-linear elements in the steady state, time, or frequency domains. However, SPICE simulation is prohibitively time consuming and doesn't allow for rapid feedback in the design process. To illustrate this, a single-layer perceptron was simulated in both HSPICE (a SPICE simulator from Synopsys) and MATLAB, a mathematical modeling language and simulation environment that is optimized for linear algebra computations. The perceptron is a good sub-architecture to evaluate simulation time because it is an essential building block in an NMS (and most neuromorphic systems), and is compounded many times to create a large-scale system. Figure 3.12(a) shows the simulation time of a perceptron in HSPICE. A very simple current mirror synapse (discussed in the next chapter) connects each linear input neuron to a linear output neuron. A number of vectors between 1 and 1000 were applied to the inputs (which were either all '0's or all '1's) for perceptrons of different sizes. From the plot, the simulation time appears to be superlinear in both N and m. More precisely, the simulation time will grow with the switching activity and the number of circuit elements present because each time the voltage or current at a particular node is changed, the SPICE simulator has to reevaluate device models and solve nodal equations. Indeed, it has been shown that neural network simulation times in SPICE grow almost quadratically with the size of the network [84]. Figure 3.12(b) shows the same perceptrons simulated in MATLAB, where the output is simply given by $y = \mathbf{u} \cdot \mathbf{w}$. Notice that the simulation time is several orders of magnitude lower than those from HSPICE. There is still a linear dependence on the number of input vectors. This is because MATLAB has to evaluate a new dot product for each input. Importantly, however, the simulation time has a weak sublinear dependence on the size of the perceptron. This is critical because, for a given problem, the number of training/test patterns applied to an NMS is usually fixed, while the size of the NMS will grow to achieve better performance. The speedup (calculated by dividing the HSPICE simulation time by the MATLAB simulation time) is shown in Figure 3.13. For small perceptron sizes, the speedup drops off linearly with the number of input vectors. This is because each dot product is computed in the body of a `for` loop. Since MATLAB is an interpreted language the time spent evaluating the `for` loop code is comparable to the time spent in the body when N is small. However, as N becomes large, the dot product evaluation takes the majority of the time, and the speedup curve starts to flatten out. A speedup of $\approx 10^5$ is observed for N=1000, m=1000, which is a modest size, considering that each NMS will have more than 1 perceptron, and each input pattern will be evaluated multiple times.

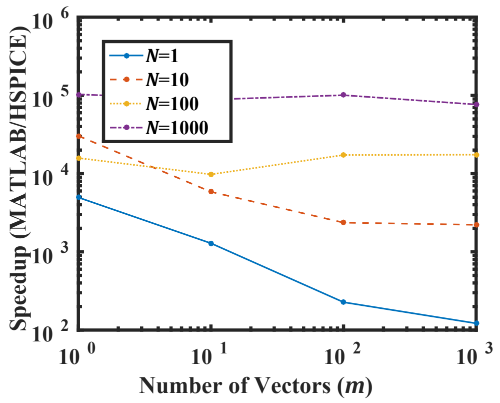


Figure 3.13: Speedup of MATLAB simulation compared to HSPICE for a single-layer perceptron.

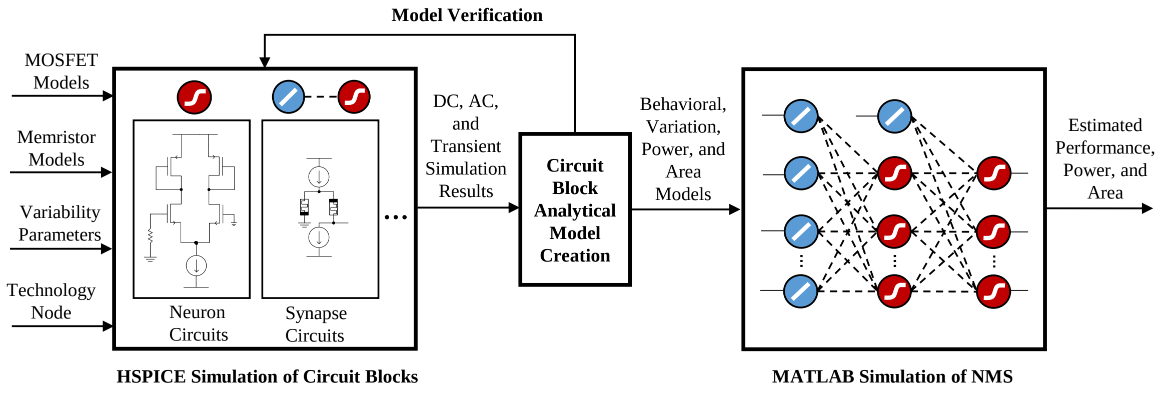

Model Verification
MOSFET Models
Memristor Models
Variability Parameters
Technology Node
Neuron Circuits
Synapse Circuits
HSPICE Simulation of Circuit Blocks
DC, AC, and Transient Simulation Results
Circuit Block Analytical Model Creation
Behavioral, Variation, Power, and Area Models
MATLAB Simulation of NMS
Estimated Performance, Power, and Area

Figure 3.14: NMS simulation strategy, where HSPICE is used for circuit-level design and analysis, and MATLAB is used for system-level simulation.

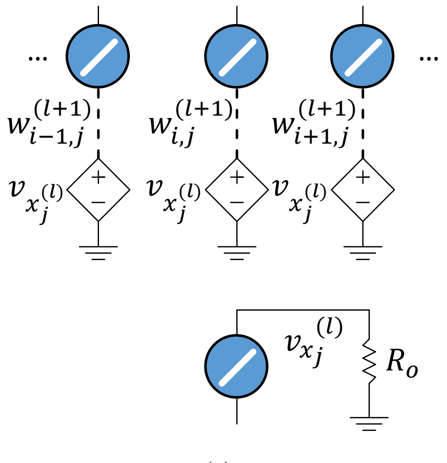
In order to take advantage of this large speedup, this dissertation adopts a simulation strategy that makes use of both HSPICE and MATLAB (Figure 3.14). Small circuit blocks are designed and simulated using HSPICE. These include neurons, synapses, and other neuromemristive primitives, which are discussed in the next chapter. Then, the simulation results are used to create analytical models of the circuits' behavior, variability, power consumption, and area. These models are checked against the HSPICE simulations and adjusted to maximize their accuracy. Once the models have been verified, they are integrated into a MATLAB toolbox developed in this dissertation, which is used for all system-level simulations. The advantages of this approach are that it reduces NMS simulation time by several orders of magnitude and it affords the user the convenience of the MATLAB environment, which easily integrates with computer vision toolboxes. The obvious drawback is that there will be some loss of accuracy when approximating circuit behavior. In addition, the behavioral models developed will not include any information about timing and transient behavior, so one has to rely on rough estimates to measure overall latency.

3.4 Current-mode Designs and Analog Signal Representation

So far, this chapter has outlined the device models and simulation strategy that are used in this work. In the next chapters, those models and simulation techniques will be applied to the design of synapse, neuron, and plasticity circuits for NMSs. However, there is one important design choice that must be made before continuing with the design of primitive circuits: Should NMSs perform computations on information represented by currents (current-mode), voltages (voltage-mode), or a combination of the two? Current-mode design techniques date back to 1975, when Gilbert proposed a general class of circuits composed of devices (called translinear elements) that have an exponential current-voltage relationship [85]. Gilbert's *translinear principle* states that circuits configured with loops of translinear elements (translinear circuits) behave in a very predictable way: The products of the currents flowing in one direction equal the products of the currents flowing in the other direction. Initially, the translinear principle was demonstrated with bipolar junction transistor (BJT) devices, but it is also applicable to MOS devices operating in weak inversion. Complex computations such as vector magnitude calculations can be implemented in current-mode using the translinear principle with a handful of transistors. There is no similar design principle for voltage-mode circuits.

Current-mode circuits have several other advantages over voltage-mode designs. They are generally able to operate at lower supply voltages and typically can achieve higher bandwidths, sometimes approaching the MOSFET intrinsic frequency f_T [86]. In addition, current representations of information have an inherent advantage in terms of communication. When voltages are sent along long routing paths, they incur losses due to series resistances, diminishing the integrity of the signal. Biology's solution to this problem has been to send long-range communications in the form of spikes which are regenerated along myelenated axons. However, it is still largely unknown how neural information is encoded in spikes and rate encoding is still the dominant scheme used in hardware implementations of spiking networks. It is often easier to represent spiking rates in hardware as continuous analog values, albeit with some reduced noise tolerance. However, buffering analog voltages requires expensive hardware, with carefully-designed amplifier circuits (e.g. common drain amplifiers) to achieve unity gain. In addition, simple analog voltage buffers typically operate in small-signal operating regions and require very careful biasing to obtain zero offset. Better designs typically employ a source follower op-amp configuration which can handle rail-to-rail input and output signals. However, even the simplest op-amps consisting of differential and gain stages require 7 transistors. Contrast that with current-mode designs, which can communicate information over long distances with relatively little signal degradation.

In addition to the general advantages of current-mode circuits, there are also specific benefits when these designs are used to implement neuromemrisitive architectures and systems. Consider the two configurations in Figure 3.15. In the first case (Figure 3.15(a)),



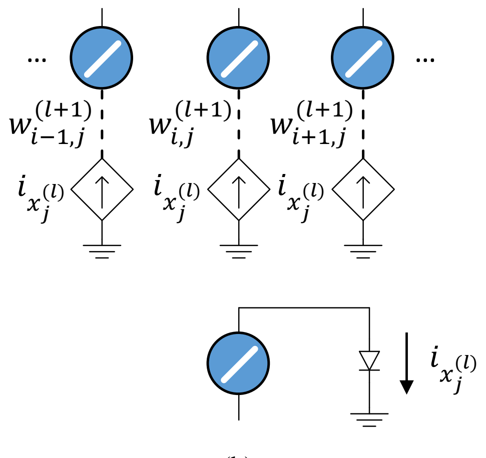


Figure 3.15: (a) Voltage-mode NMS, where neuronal activations are represented as voltages and synapses typically operate via transconductance. (b) Current-mode NMS where neuronal activations and the results of synaptic weighting are represented as currents.

a pre-synaptic neuron has a voltage output $v_{x_j^{(l)}}$ which falls across an output impedance R_o connected to a small-signal ground. Here, each neuron is modeled as an ideal voltage source that implements a linear activation function. However, the pre- and post-synaptic neurons may have generally have any activation function. The gain of the pre-synaptic neuron can be described as $A_v = g_m R_o$, where g_m is a transconductance factor. Now, consider the direct connection of the pre-synaptic neuron to all of the outgoing synapses, which can be characterized by several parallel conductance values. In this case, the gain of the pre-synaptic neuron becomes $A_v = g_m \left(R_o || \cdots \frac{1}{G_{i-1,j}^{(l+1)}} || \frac{1}{G_{i,j}^{(l+1)}} || \frac{1}{G_{i+1,j}^{(l+1)}} \cdots \right)$. Therefore, if the neuron's fanout is high, then small weight values in the outgoing synapses (typically represented as high conductances) will significantly diminish the neuron gain. Therefore, it is usually best practice to add a buffer before each outgoing synapse to reduce the loading effect on the pre-synaptic neuron. In fact, this technique is analogous to biological nervous systems, where each outgoing synapse is buffered using synaptic vesicles held within the pre-synaptic terminal (synaptic bouton). As discussed earlier, buffering voltage-mode analog neurons is expensive in terms of hardware area. In contrast, a current-mode design (Figure 3.15(b)) affords the ability to buffer pre-synaptic neuron outputs using current-controlled current sources (typically simple CMOS current mirrors), which have smaller area and power requirements. One voltage-mode design that has overcome the loading issue is the memristor bridge synapse proposed by Kim et al. [39, 51, 87, 88]. The design uses complementary memristive devices in series combinations such that the total synapse input resistance is always high, regardless of the weight state. However, the design is complex, requiring 4 memristive devices and 3 MOSFETs for each synapse circuit.

Although the current-mode design approach is attractive, there are some challenges to consider when designing NMSs using current-mode circuits. First, a current can't be distributed through multiple circuit branches without buffering. Second, since current mirrors are employed extensively, current-mode designs are especially prone to mismatch-related problems. The effects of mismatch on the circuit, architecture, and system-level performance are studied extensively in this work.

3.5 Summary

This chapter provided a detailed discussion on the device models (MOSFETs and memristors), simulation strategy, and design methodology adopted in this work. The key contributions and results from this chapter are:

- Semi-empirical MOSFET (45 nm high V_{th0}) and memristor (Ag chalcogenide and CBRAM) models were designed to capture essential device behavior while minimizing model complexity. Key memristor properties that can be improved for future work are the *on* resistance, which should be increased for easier programming, and the conductance ratio g, which should also be increased to improve sense margins.

- The device models designed in this work can be simulated in SPICE or MATLAB. While SPICE provides more detailed simulation results (especially transient behavior), MATLAB simulations yield orders-of-magnitude reduction in simulation time.
- MOSFET and memristor devices will be integrated, primarily, into subthreshold current-mode circuits for primitive NMS operations. The current-mode approach has several advantages, including reduced design complexity, improved input/output range, etc. There are also some challenges to consider with a current-mode design, including device mismatch, which is exacerbated in subthreshold.

Chapter 4

Synapse and Neuron Circuits

This chapter presents the synapse and neuron circuits designed in this work. Synapses were designed for constant weight values, random weight values, and adjustable weight values. Careful attention is given to minimizing the synaptic area. This is critical, since the number of synapses in an NMS (or any neural network) grows quadratically with the number of neurons. Previous synapse designs have assumed ideal memristor behavior, where switching is deterministic, and continuous [33–40, 89]. For example, many designs have been simulated using a linear ionic drift memristor model [19], characterized by a smooth hysteresis curve. However, memristor switching behavior is usually discontinuous, indicating the devices can only achieve a small set of conductance states. Furthermore, the exact values of these states and the required conditions (e.g. write voltage) have high variability. This work captures memristive circuit behavior that accurately reflects the experimental characteristics of the devices. The neuron designs presented in this chapter include common activation functions (e.g. sigmoid, linear, and threshold), as well as periodic and rectified linear. Behavioral, power, area, and variation models are presented for each circuit. In later chapters, these models are integrated into system-level simulations of NMSs for visual information processing.

4.1 Synapse Circuits

4.1.1 Constant Current Mirror Synapse

4.1.1.1 Basic Operation

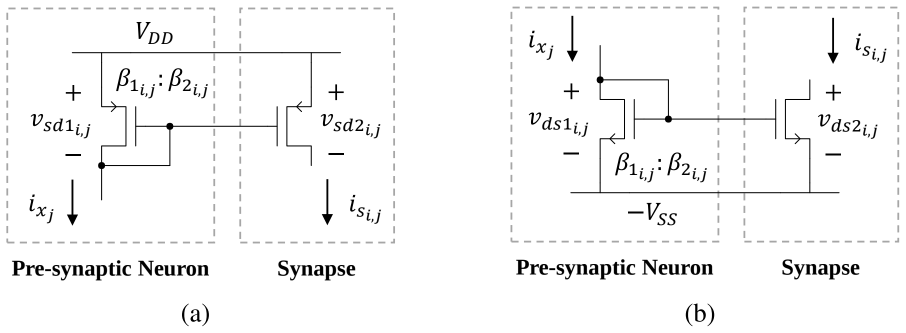


Figure 4.1: Constant current mirror synapse circuits for (a) positive weights and (b) negative weights.

NMSs often employ synapses with constant and sometimes random weight values. Two network topologies where these are encountered are MLPs with random hidden layer weights [90] and reservoirs [91]. In these topologies, constant and random weights are used to map data from lower to higher dimensional spaces and vice versa. Additionally, they sometimes form feedback connections in recurrent networks. One approach to design constant weights is to use a CMOS current mirror, as shown in Figure 4.1. The synapse is the output MOSFET, while the input MOSFET is part of the pre-synaptic neuron. The weight value is defined as

$$w_{i,j} \equiv \frac{i_{s_{i,j}}}{i_{x_j}} = \frac{\beta_{2_{i,j}}}{\beta_{1_{i,j}}}\exp\left(\frac{\Delta V_{th0}}{nV_T}\right)\frac{\Lambda\left(v_{ds2}, L_2\right)_{i,j}}{\Lambda\left(v_{ds1}, L_1\right)_{i,j}} = w'_{i,j}\Pi_{i,j}, \tag{4.1}$$

The ratio of Λ functions occurs frequently, so it has been given the designation $\Pi_{i,j}$. If both

MOSFETs in the mirror have the same length and the same v_{ds} value, then the synaptic weight $w'_{i,j}$ will be the ratio of the current factors $\beta_{2_{i,j}} : \beta_{1_{i,j}}$ of the two MOSFETs (the gain of the current mirror), which is controlled via sizing. PMOS mirrors allow the synapse to excite the post-synaptic neuron (positive weights), whereas NMOS mirrors inhibit the post-synaptic neuron (negative weights).

There are several important design aspects related to the synapse circuits in Figure 4.1. First, the maximum or minimum voltages at the output nodes are equal to $V_{DD} - V_{ds_{sat}}$ and $-V_{SS} + V_{ds_{sat}}$, respectively, where $V_{ds_{sat}} \approx 100$ mV. Second, the inputs to the synapses are unipolar, so

$$s_{i,j} = H\left(x_j\right) w_{i,j} = H\left(x_j\right) w'_{i,j}\Pi_{i,j} \tag{4.2}$$

where $H\left(\cdot\right)$ is the Heaviside step function, $v_{ds1_{i,j}}$ is the drain-source (source-drain for PMOS) voltage of the input MOSFET and $v_{ds2_{i,j}}$ is the drain-source voltage for the output MOSFET. Note that this equation is approximate, because $s_{i,j} \neq 0$ when $x_j =\leq 0$. Rather, it will be bounded by the leakage currents of the MOSFETs.

It is also important to consider the case where the v_{ds} values are different for each MOSFET in the mirror. For a given current flowing through the input transistor, Equation (3.8) can be used to find the value of $v_{ds1_{i,j}}$. If $v_{ds2_{i,j}}$ is constant, then (3.8) can be calculated immediately. In general, however, $v_{ds2_{i,j}}$ will vary as a function of the synaptic current. This means that, especially for small channel lengths, $|w_{i,j}|$ will not be constant, as it is typically assumed to be in artificial neural networks. To reduce the effect of the Λ ratio in (4.2), one can increase the channel lengths. For example, just doubling the channel lengths (i.e. from 45 nm to 90 nm) will significantly reduce the dependence of the weight values on variations in v_{ds}. Another alternative would be use a current mirror with higher output impedance, such as a cascode configuration. Both options, however (doubling the channel length and using cascode mirrors with minimum channel length) will double the synaptic area. In addition, the use of a cascode configuration results in additional MOSFETs between the supply rails, reducing (increasing) the maximum (minimum) synaptic input and output voltages.

4.1.1.2 Area and Power

The area of a constant current mirror synapse is equal to the area of the output MOSFET, $W_{2_{i,j}} L_{2_{i,j}}$, which will depend on the synaptic weight. Let $A^*_{cs_{i,j}}$ be defined as

$$A^*_{cs_{i,j}} = A_{min} \frac{w'_{i,j} \left(W_{1_{i,j}} / L_{1_{i,j}}\right) L_{2_{i,j}}}{A_{min}}, \tag{4.3}$$

where A_{min} =45 nm $\times$ 45 nm is the minimum area of a transistor. Now, the synaptic area can be expressed as

$$A_{cs_{i,j}} = A^*_{cs_{i,j}} \left\lceil \left\lceil \frac{A^{min}_{cs}}{A^*_{cs_{i,j}}} \right\rceil^{1/2} \right\rceil^2, \tag{4.4}$$

where A^{min}_{cs} is a minimum area constraint. Equation (4.4) warrants some explanation. If $A^{min}_{cs} \leq A^*_{cs_{i,j}}$, then $A_{cs_{i,j}} = A^*_{cs_{i,j}}$. Otherwise, the area increases to meet the minimum area requirement. While increasing the area, one must maintain the same $\beta_{2_{i,j}}$ to keep $w'_{i,j}$ constant. This amounts to increasing both the length and width by the same integer factor, which increases the area by the square of that factor.

The power consumption of the constant current mirror synapse can be approximated at low frequencies as

$$P_{cs_{i,j}}^{avg} \approx \eta_j w_{i,j} I_{max} V_{DD}, \tag{4.5}$$

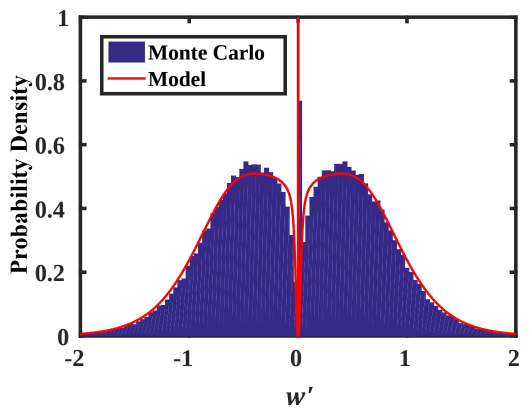
where η_j is the activity factor of pre-synaptic neuron. A value of $\eta = 0.5$ is usually a good estimate.

4.1.1.3 Process Variations

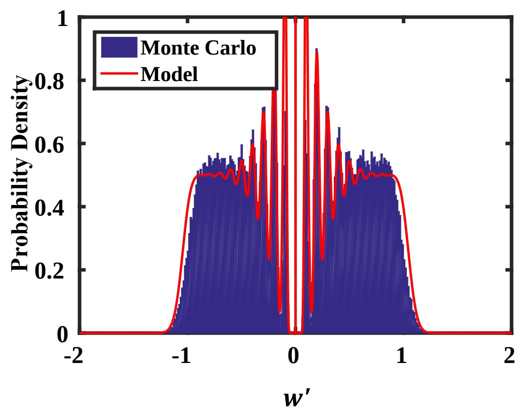
Variations in threshold voltage will have a large impact on most of the NMS primitive circuits. For the constant current mirror synapse, the effect will be reflected in the distribution of $w'_{i,j}$, defined in (4.1). Recall that ΔV_{th0} is a Gaussian-distributed random variable with zero mean and variance given by (3.9). In addition, any desired (discrete) distribution may be chosen for the current factors by sizing transistors appropriately. The probability density function for $|w'|$ becomes

$$f\left(|w'|\right) = \begin{cases} \sum\limits_{\phi} \Pr\left(\Phi = \phi\right) \frac{\beta_1}{\beta_2} \ln\mathcal{N}\left(|w'|\frac{\beta_1}{\beta_2}; 0, \frac{\sqrt{\frac{1}{2}\frac{A_{V_{th0}}^2}{W_1 L_1} + \frac{1}{2}\frac{A_{V_{th0}}^2}{W_2 L_2}}}{nV_T}\right), & |w'| > 0 \\ \Pr\left(|w'| = 0\right) \delta\left(|w'|\right), & |w'| = 0 \end{cases} \tag{4.6}$$

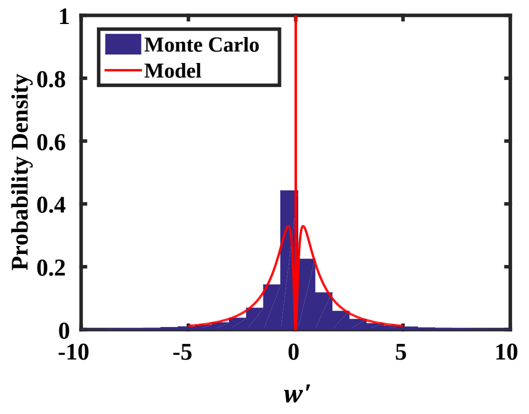
where Φ is a random set of W_1, L_1, W_2, and L_2, and ϕ is a particular value of Φ. In addition, $\ln\mathcal{N}\left(x; \mu, \sigma\right)$ is the log-normal PDF with μ and σ being the mean and standard deviation of the associated normal distribution. Finally, $\delta\left(\cdot\right)$ is the Dirac delta function. The distribution represented by (4.6) can be derived based on Rohatgi's result for the distribution of products of random variables [92]. The delta function accounts for the fact that weight values equal to 0 are represented by removing the synaptic connection between a pre- and post-synaptic neuron, so it no longer makes sense to define the distribution in terms of transistor geometry and threshold voltage variation when $|w'|$ =0.



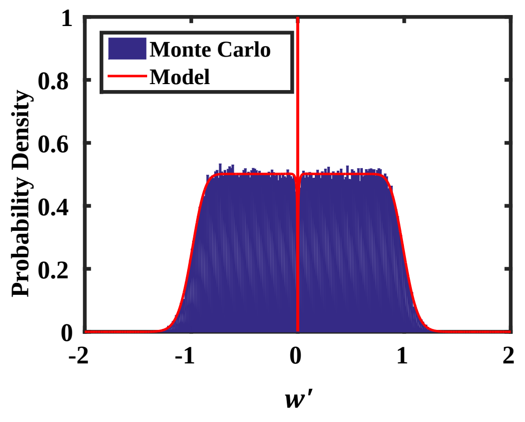
(a)



(b)



(c)



(d)

Figure 4.2: Probability densities for w' in the constant current mirror synapse. (a) W_1/L_1 =10/1, $A_{cs}^{min} = A_{min}$. (b) W_1/L_1 =10/1, $A_{cs}^{min} = 25A_{min}$. (c) W_1/L_1 =1/1, $A_{cs}^{min} = A_{min}$. (d) W_1/L_1 =100/1, $A_{cs}^{min} = A_{min}$. In all cases W_2/L_2 is drawn from a discrete uniform distribution between 1 and W_1/L_1. Zero-valued weights are also added with probability $1/(1 + W_1/L_1)$.

The effects of threshold variation on constant current mirror weight distributions are shown in Figure 4.2. Each plot shows the results of 100,000 Monte Carlo simulations (performed in HSPICE) for both excitatory and inhibitory synapses, resulting in a total of 200,000 samples. For each sample, W_1, L_1, and L_2 were held constant while W_2 and

ΔV_{th0} were drawn from a discrete uniform distribution and a continuous Gaussian distribution, respectively. Specifically, W_2 varied such that β_2/β_1 followed a discrete uniform distribution between L_1/W_1 and 1. In addition, zero-valued weights were added with probability $1/(1+W_1/L_1)$, which modified β_2/β_1 to be drawn a uniform distribution between 0 and 1 a resolution of L_1/W_2. In Figure 4.2(a), W_1/L_1 =10/1 and $A_{cs}^{min} = A_{min}$. The normally-distributed threshold variation changes the shape of the distribution of $|w'|$ from discrete uniform (what would be expected if $A_{V_{th0}}$ =0) to a continuous distribution with long tails. This enables very high resolution weights at a much lower area cost than might be predicted by (4.4). Based on the fact that the variance is related to the inverse of the transistor area, one might expect to tighten the distribution by increasing the minimum synapse transistor area A_{cs}^{min}. Figure 4.2(b) shows the case where W_1/L_1 =10/1 and A_{cs}^{min} has been increased to A_{min}. We see two effects. As expected, the distribution tightens with very low density outside of the [-1,1] domain. In addition, one notices the formation of peaks around the values $\pm\frac{1}{10}, \pm\frac{2}{10}, \ldots$, because the continuous distribution approaches the discrete distribution (where $A_{V_{th0}}$ =0) as the transistor areas increase. It's also possible to use minimum sizing for all of the MOSFETs in the synapse circuit, resulting in the distribution shown in Figure 4.2(c). Notice that the tails of the distribution are very long. While using minimum sizing does reduce the synapse area, (4.5) reveals that it will also increase the average synaptic power consumption. Therefore, there is a three-way tradeoff between the distribution shape, the synapse area, and the synaptic power consumption. Figure 4.2(d) shows a fourth case where W_1/L_1 =100 and $A_{cs}^{min} = A_{min}$. Observe the difference between Figures 4.2(d) and 4.2(b). In both cases, the average synaptic area has been increased, resulting in a tightening of the distribution. However, there are no prominent peaks (except for the one at $|w'|$ =0) in Figure 4.2(d). The reason is that, for larger weight values (e.g. $|w'|\approx$1), the large variance resulting from the β_2/β_1 ratio is cancelled by the small variance from the larger synapse area. The opposite is true for smaller weight values. This is also the reason that the peaks in Figure 4.2(b) are large near the center of the distribution and smaller moving outward. Until now, we have made the tacit assumption that synapses do not share pre-synaptic neurons. In practice, however, each neuron will drive several post-synaptic neurons through synaptic connections. In Figure 4.1, this amounts to multiple output MOSFETs being driven by the same diode-connected input MOSFET. Consequently, every set of synapses belonging to a particular pre-synaptic neuron will have its own distribution of weights, which could be problematic. For example, consider the simple case where an NMS has a single input neuron, driving several hidden-layer neurons through constant current mirror synapses. If we are lucky, the threshold voltage of the input MOSFET will fall somewhere in the middle of the distribution. However, if the threshold voltage falls at an extreme value of the distribution, then all of the magnitudes of the synaptic weights will be either very large or very small. In other words, the mean of the distribution depends on particular value of the threshold voltage of the input MOSFET, which is reflected in the following conditional distribution:

$$f\left(|w'|\middle|V_{th01}\right) = \begin{cases} \sum\limits_{\phi} \Pr\left(\Phi = \phi\right) \frac{\beta_1}{\beta_2} \ln\mathcal{N}\left(|w'|\frac{\beta_1}{\beta_2}; \frac{V_{th0p(n)} - V_{th01}}{nV_T}, \frac{\sqrt{\frac{1}{2}\frac{A^2_{V_{th0}}}{W_2 L_2}}}{nV_T} \right), & |w'| > 0 \\ \Pr\left(|w'| = 0\right) \delta\left(|w'|\right), & |w'| = 0 \end{cases} \tag{4.7}$$

Figure 4.3 shows distributions of w' from an excitatory synapse with different areas for

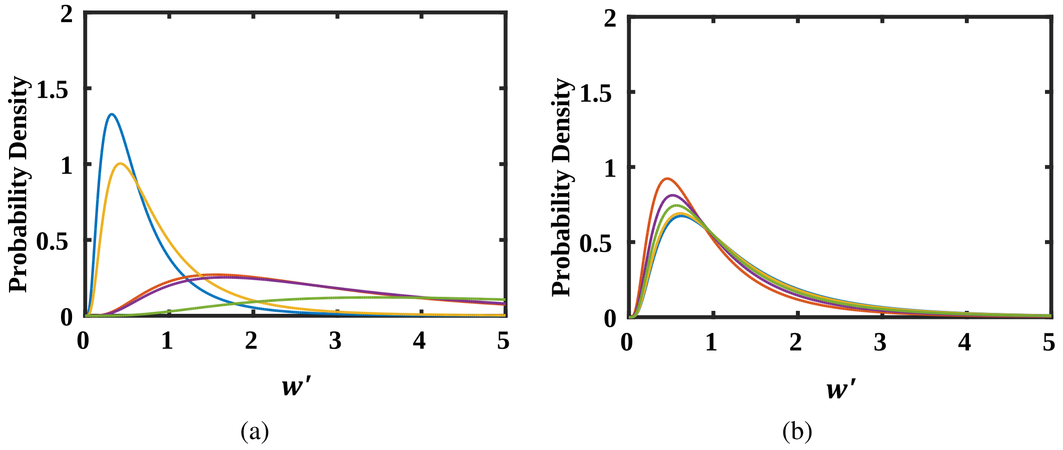


Figure 4.3: Conditional distributions of w' for excitatory synapses connected to different pre-synaptic neurons with $W_2L_2 = A_{min}$. (a) $W_1L_1 = A_{min}$ resulting in a large variation in the distributions' means. (b) $W_1L_1 = 10A_{min}$, resulting in reduced variation of the distributions' means.

the input MOSFET. In Figure 4.3(a), $W_1L_1 = A_{min}$ and $W_2L_2 = A_{min}$, resulting in a large variation in the distributions means. In contrast, Figure 4.3(b) shows results for $W_1L_1 = 10A_{min}$ and $W_2L_2 = A_{min}$, which yields a tighter distribution of the means. Therefore, one can reduce pre-synaptic neuron-dependent bias, by increasing the size of the input MOSFET.

As a final note, it is important to remember that MOSFETs were assumed to be closely spaced in the variation analysis, allowing one to neglect the spacing-dependent variation of threshold voltage [62]. In fact, this assumption is used for all variation analyses in this work. However, as a pre-synaptic neuron's fanout increases, the spacing between the input and output MOSFETs will also increase, making the assumption less valid. One solution may be to have each pre-synaptic neuron drive multiple subsets of synapses though dedicated input MOSFETs, each of which is sized large enough to ignore the effect of threshold voltage variation. However, it would be very difficult to determine the number of subsets required to keep the closely spaced assumption valid, and would likely require statistical data from fabricated test chips.

4.1.1.4 Constant Current Mirror Synapse with Bipolar Input

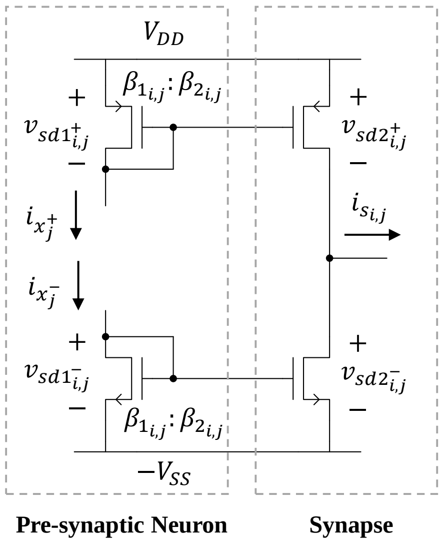


Figure 4.4: Constant weight synapse circuit with bipolar input.

The constant current mirror synapse, as it has been presented so far, has a unipolar input, making it incompatible with pre-synaptic neurons that have bipolar activation functions, such as $\tanh$. However, the design can be modified to accommodate bipolar inputs, provided that the pre-synaptic activation function can be expressed as a difference of positive and negative components, $x_j = x_j^+ - x_j^-$. The circuit schematic is shown in Figure 4.4. The circuit is essentially a combination of the excitatory and inhibitory constant weight synapses, where the excitatory part is driven by the positive component of the pre-synaptic neuron's output, and the inhibitory part is driven by the negative component of the pre-synaptic neuron's output. The output is expressed as

$$s_{i,j} = x_j^+ w'^+_{i,j} \Pi^+_{i,j} - x_j^- w'^-_{i,j} \Pi^-_{i,j}. \tag{4.8}$$

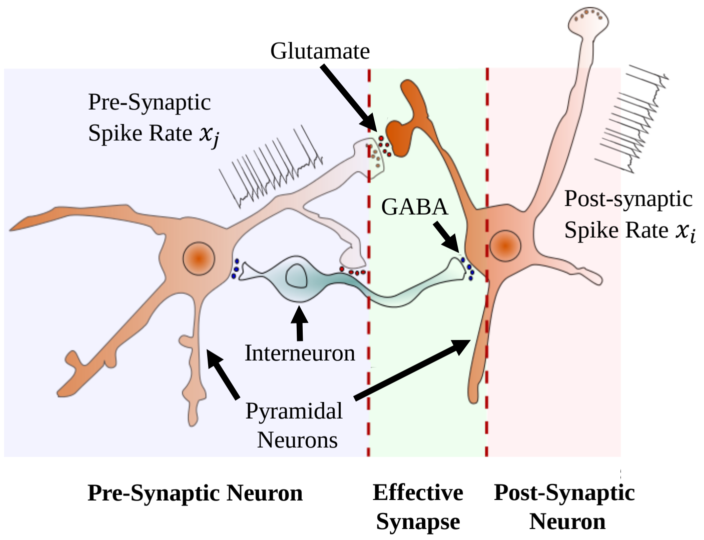
Of course, if the effects of v_{ds} are ignored and it is assumed that $w'^{+}_{i,j} = w'^{-}_{i,j} = w_{i,j}$, then (4.8) just becomes $s_{i,j} = x_j w_{i,j}$.

The area and power of the synapse in Figure 4.4 will be double the area and power of the unipolar input synapse. In addition, the effect of threshold voltage variation can be modeled by following the same approach as in the unipolar case.

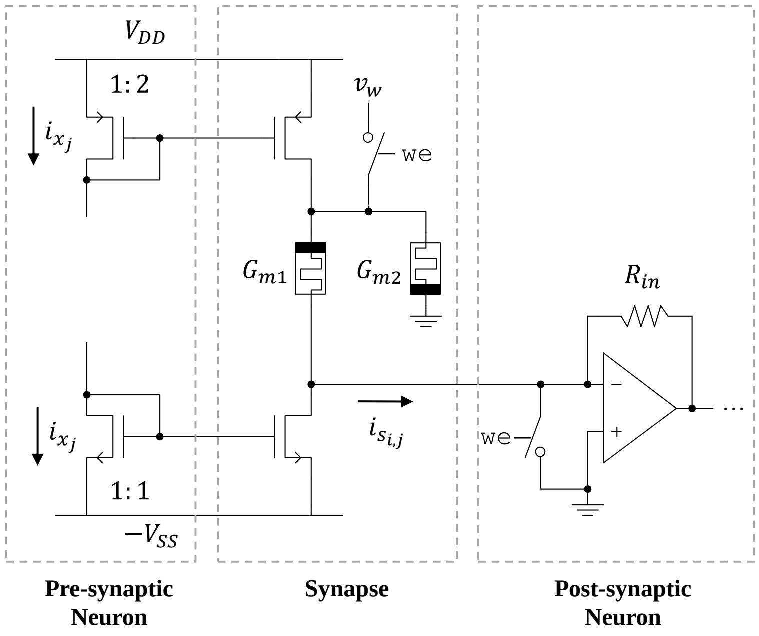
4.1.2 Bipolar Weight Memristive Synapse

4.1.2.1 Basic Operation

In addition to the constant weight synapses presented above, an NMS requires adjustable synapses to facilitate learning. Furthermore, it is advantageous to design synapse circuits that can have bipolar weight values. The reason is that neural networks composed of bipolar weights can generally separate and fit data better than those that have unipolar weight values, especially with little or no input pre-processing. To this end, a current-mode memristive synapse was designed to achieve adjustable bipolar weight values. The design is inspired by the cortical microcircuit shown in Figure 4.5(a). Here, a post-synaptic pyramidal neuron is driven by both excitatory (glutamatergic) and inhibitory (GABAergic) synapses. The relative strength of the two synaptic connections determines an effect weight. The circuit schematic is shown in Figure 4.5(b). The synapse's input current is the output current of the pre-synaptic neuron. Notice that both the diode-connected PMOSFET and the diode-connected NMOSFET from the pre-synaptic neuron are used to mirror the input in two places. The PMOS mirror has a 1:2 size ratio, so the output of the mirror is $2i_{x_j}$. The



(a)



(b)

Figure 4.5: Bipolar weight memristive synapse. Two anti-parallel memristors control the relative ratio of excitation to inhibition at the output.

memristors in the circuit are in parallel, since they share a common top node, and their bottom nodes are both at 0 V. The memristor m_1 is connected to the input of a post-synaptic neuron. Notice that the input of this neuron is a virtual ground. Assuming this synapse connects a j^{th} pre-synaptic neuron to an i^{th} post-synaptic neuron, then its output $s_{i,j}$ can be described as (assuming infinite opamp gain).

$$s_{i,j} = 2x_j \Pi^{+}_{i,j} \frac{G_{m1}}{G_{m1} + G_{m2}} - x_j \Pi^{-}_{i,j} \tag{4.9}$$

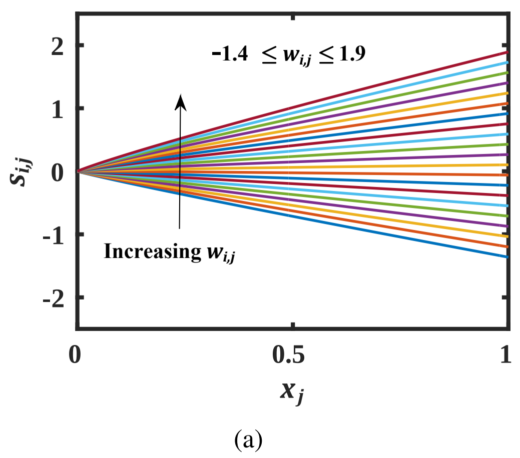
where G_{m1} and G_{m2} are the conductances of the two memristors. Therefore, the synaptic weight is given by

$$w_{i,j} = 2\Pi^{+}_{i,j} \frac{G_{m1}}{G_{m1} + G_{m2}} - \Pi^{-}_{i,j}. \tag{4.10}$$

If the effects of the drain-source voltages are negligible, then this becomes

$$w'_{i,j} = 2\frac{G_{m1}}{G_{m1} + G_{m2}} - 1. \tag{4.11}$$

When both memristors have a high g ratio, then $w'_{i,j}$ will range approximately from -1 to +1. Figure 4.6 shows the synaptic output characteristics for circuits with two different values of the channel length L. In Figure 4.6(a), $L = 45$ nm, resulting in an increased range of the synaptic weight value to [-1.6, 2.2]. Despite this increased range, the synapse's linearity is relatively unaffected by effects of drain-source voltages. The reason is that both $\Pi^{+}_{i,j}$ and $\Pi^{-}_{i,j}$ are approximately constant. This stems from i.) the fact that the drain-source voltage of the NMOS transistor is constant (because the drain is at a virtual ground), and ii.) the small currents flowing through the memristors cause only very small changes in the PMOS transistor's drain voltage. The effects of the drain-source voltages are diminished further when the channel length is increased, as shown in Figure 4.6(b). Here, L is increased by a factor of 4, resulting in a synaptic weight range close to the ideal one. Of course, the question becomes whether or not it matters if the weight range is increased (as in the case



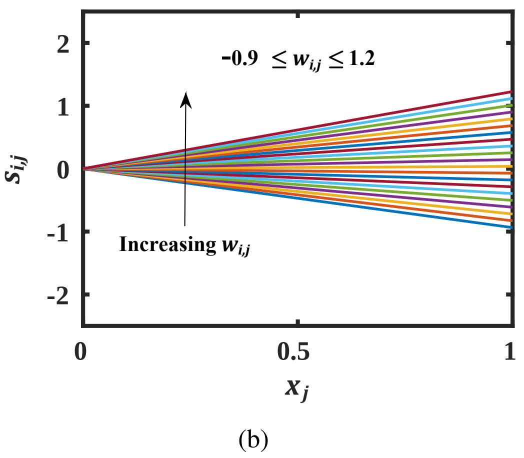


Figure 4.6: Bipolar weight memristive synapse output. (a) $L = 45$ nm, (b) $L = 180$ nm.

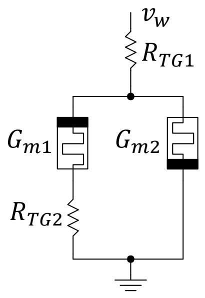


Figure 4.7: Equivalent write circuit for the bipolar weight memristive synapse.

of $L = 45$ nm). Since the synapse shows excellent linearity, its behavior can be modeled its behavior as a constant factor $w_{i,j}$ that multiplies the pre-synaptic output, with a range that is determined from SPICE simulations.

It is critical that the states of the memristors are not changed unintentionally. To ensure this, the maximum quotient of the memristor current and memristor conductance should be well below the threshold voltage. Since the maximum current value is on the order of 10^9, and the minimum conductance is on the order of 10^{-5}, the voltage across the memristors will nominally be in the 100s of microvolts range, which is 3 orders of magnitude below the threshold voltages.

Modification of the synaptic weight value is accomplished through two switches in the synapse and post-synaptic neuron. The switch is controlled by a write enable signal `we`, which allows a write voltage v_w to be applied to the memristors. Notice that the memristors are anti-parallel, so application of a positive write voltage will increase G_{m1} and decrease G_{m2}, while the application of a negative write voltage will decrease G_{m1} and increase G_{m2}. A second switch, which is part of the post-synaptic neuron, creates a strong connection to ground at the negative terminal of the first memristor. Both switches are implemented using transmission gates. A critical design consideration stems from the large ON resistance of the 45 nm MOSFETs used in this work. During a write operation, if v_w is not large enough, then the series resistances (see Figure 4.7) of the switches will cause the voltage across the memristors to be below their thresholds. In turn, the weight of the synapse will not be modifiable. This is a critical point, which has often been missed in the literature, where switches are frequently modeled with zero impedance. Figures 4.8(a) and 4.8(b) show the transition of the synaptic weight from -1 to 1 and 1 to -1, respectively. In both cases, a 1 μs, 3.5 V pulse was applied repeatedly, causing the memristors' state variables to transition. Both transitions take $\approx 10^3$ pulses to complete. The transition rate is directly related to the learning rate α, and can be controlled via the applied pulse width. Shorter pulse widths correspond to smaller α values and larger pulse widths correspond to larger α values. Furthermore, from (3.18), it can be seen that $\Delta\gamma$ is linear in Δt, so it will be assumed that $\Delta w_{i,j}$ is also linear in the applied pulse width.

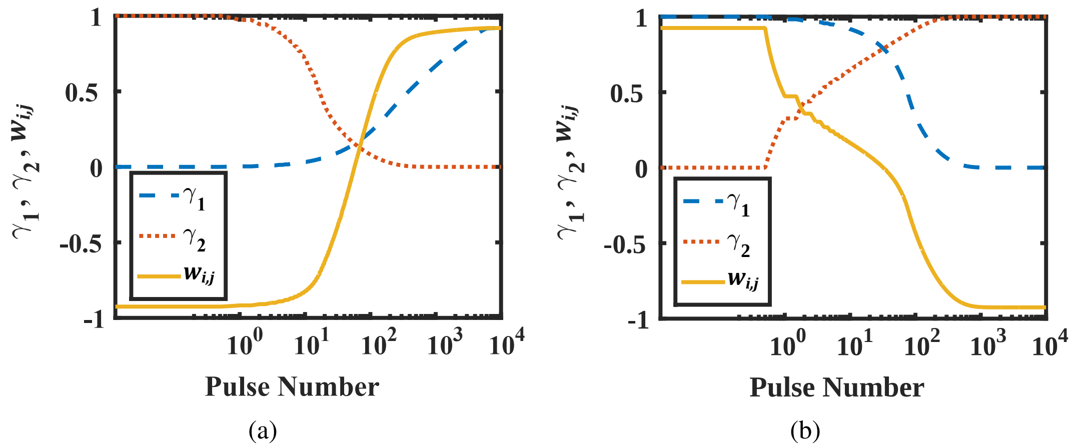


Figure 4.8: Evolution of the weight in the bipolar weight memristive synapse. (a) The weight is changed from -1 to 1 by applying positive write pulses. (b) The weight is changed from 1 to -1 by applying negative write pulses. In both cases, $|v_w| =$ 3.5 V, and the write pulse width is 1 μs.

4.1.2.2 Area and Power

The bipolar weight memristive synapse designed in this work is composed of 2 MOSFETs, 2 memristors, and a transmission gate, making the total area

$$A_{ms} \approx 3A_{ms}^{min} + 2A_{TG} + 8F^2, \tag{4.12}$$

where A_{ms}^{min} is a minimum area constraint for the two current mirror output transistors, A_{TG} is the area of the transmission gate transistors, and F is the wire pitch. The $8F^2$ term is not necessarily needed, since the memristors are fabricated as a back end of line process, above MOSFETs. Compared to the memristor bridge synapse proposed by Kim et al. [93], the memristive synapse designed in this work requires 2 fewer memristors and 1 less MOSFET. If $A_{ms}^{min} = A_{min}$, then this translates to a 38% reduction in area. However, as A_{ms}^{min} becomes larger, this reduction in area becomes smaller.

The power consumption of memristive synapse designed in this work can be broken into two components. The first component is from normal signaling operation (when the weight is not being adjusted):

$$P^{avg}_{ms,signal} \approx \eta_j \left(2\Pi^{+}_{i,j} + \Pi^{-}_{i,j}\right) I_{max} V_{DD} \tag{4.13}$$

This is an order-of-magnitude approximation, since the maximum output of the pre-synaptic neuron may, in general, be larger or small than I_{max} depending on process variations. One major advantage of the proposed design over the memristor bridge synapse [93] is that the power consumption is tied to the activity of the network. In [93], there is a constant power consumption originating from a differential pair bias current. As a result, the synapse design in this work reduces power consumption by $\approx 25\%$. Although it hasn't been explored in this work, the proposed synapse design also affords the opportunity to reduce power consumption at the system level by making network activity sparse.

Another key attribute of the memristive synapse designed in this work is that its power consumption is independent of the memristors' states, which is not the case for other synapse designs proposed in the literature [33–38, 40, 41]. This independence stems from the current-mode design paradigm adopted in this work and has implications for defect tolerance. For example, consider a short-circuit defect in the second memristor in Figure 4.5(b). Since the synapse's input is a current, the defect will not affect the average power consumption. However, if the input (which is the output of the pre-synaptic neuron) was a voltage, then a short-circuit to ground could cause a large increase in the circuit's power consumption, which could cause damage to the chip.

The second component of the power consumption is from adjusting the synaptic weight:

$$P_{ms,write}^{avg} \approx D_{train}\frac{1}{2}\frac{v_w^2}{R_{TG1} + [(R_{mavg} + R_{TG2}) \parallel R_{mavg}]}, \tag{4.14}$$

where D_{train} is the duty factor for the training clock, which is in its positive phase while writing and negative phase otherwise. The mean memristor resistance, R_{mavg} is given as $R_{mavg} \equiv (R_{mon} + R_{moff})/2$.

4.1.2.3 Process Variations

The memristive synapse in Figure 4.5(b) is affected by both MOSFET and memristor variations. The MOSFET variations are modeled in a similiar way as the constant current mirror synapse, where the dominant effect is from variations in V_{th0}. To start, notice that (4.9) can be written as

$$s_{i,j} = s_{i,j}^{+\prime}\Pi_{i,j}^{+}\frac{G_{m1}}{G_{m1}+G_{m2}} - s_{i,j}^{-\prime}\Pi_{i,j}^{-} = w_{i,j}^{+\prime}x_j\Pi_{i,j}^{+}\frac{G_{m1}}{G_{m1}+G_{m2}} - w_{i,j}^{-\prime}x_{i,j}\Pi_{i,j}^{-}. \tag{4.15}$$

Here, $w_{i,j}^{+\prime}$ and $w_{i,j}^{-\prime}$ are the weight values associted with the two current mirrors, which can be modeled using (4.6) or (4.7).

Memristor variations, discussed in Section 3.2.3, also have an effect on the behavior of the synapse circuit. In particular, variations in the *on* and *off* conductance values will have a direct impact on the maximum and minimum weight values, which can be seen from (4.11). This can be especially problematic since a large fraction of trained weight values within an NMS will lie at the extrema of the weight range. This will be discussed in more detail in Chapter 7, where the effect of device variations on system-level performance is analyzed.

4.1.2.4 Crossbar Implementation

The current-mode memristive synapse described above can also be integrated into a crossbar structure as shown in Figure 4.9. Memristors in the top row inhibit, or contribute a negative component to the output, while memristors in the bottom row excite, or contribute a positive component to the output. Therefore, each crossbar column represents one component of one weight vector $\mathbf{w}_i$, which can be positive or negative. If the opamp is assumed to have high open loop gain and the wire resistances are small, then

$$x_i = \sum_{j=1}^{N} u_j^{(p)} I_{max} R \left(\frac{G_{m2} - G_{m1}}{G_{m1} + G_{m2}} \right)_{i,j}, \tag{4.16}$$

where G_{m1} and G_{m2} are the top and bottom memristors in each column, respectively. The advantages of this circuit over the stand-alone memristive synapse is that the area is reduced by one transistor, and the inputs are bipolar (i.e. the input to each synapse can be a positive or negative current). The power consumption is also reduced to

$$P_{ms,signal}^{avg} \approx \eta_j 2\Pi_{i,j} I_{max} V_{DD}, \tag{4.17}$$

which is about 2/3 of the power consumption for the non-crossbar based design. The primary disadvantage arises from unwanted current sneak paths that result from the crossbar design.

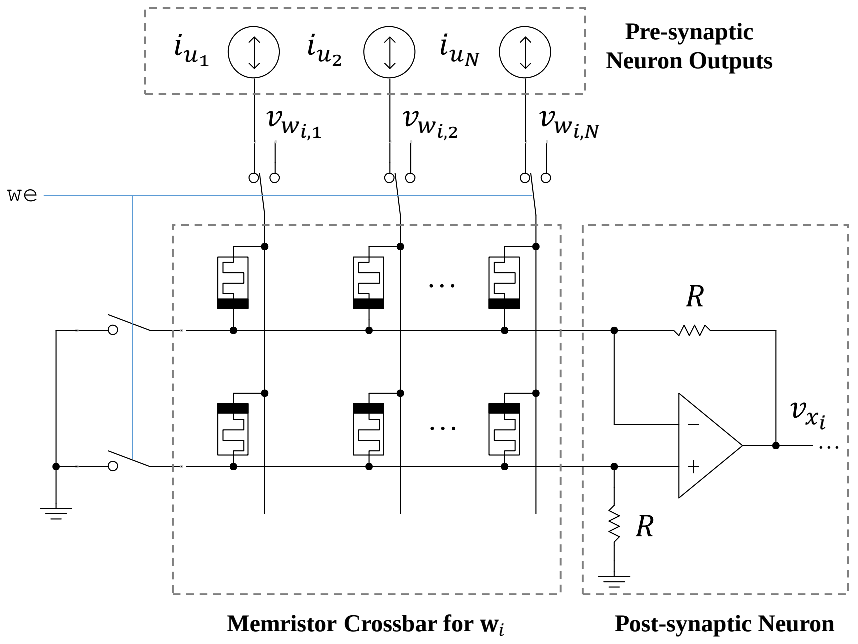


Figure 4.9: Crossbar and summing amplifier circuit for computing the distance between the input and a weight vector.

4.2 Neuron Circuits

Neurons are composed of two stages. An input stage integrates all of the signals coming from pre-synaptic neurons (convergence). The integrated signal is then passed to a second stage that applies an activation function and then distributes the new information to other neurons through outgoing synapses (divergence). This section discusses circuits and models of both of these stages.

4.2.1 Input Stages

There are two types of neuronal input stages implemented in this work. The first is an opamp input stage, where all converging synapses are connected to a virtual ground via an inverting opamp configuration. If the opamp has infinite open loop gain, then the input stage will sum the incoming signals as

$$s_i = -\sum_j w_{i,j} x_j I_{max} R_{in}, \tag{4.18}$$

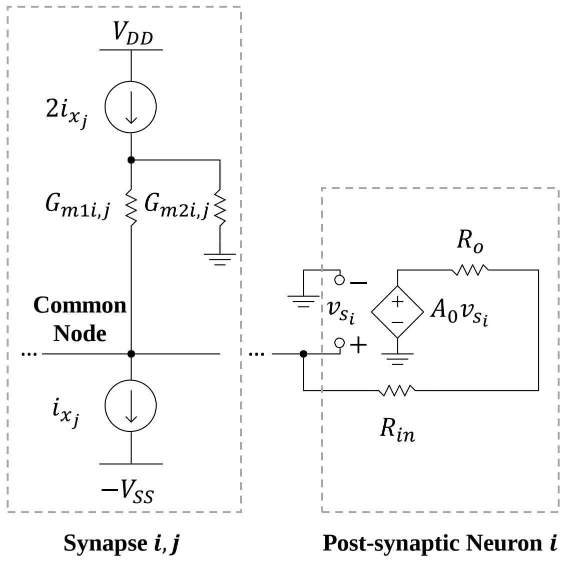
where the $-$ sign comes from the inverting opamp configuration.[1] However, practical opamp designs will not have infinite gain, so it is important to model the behavior of s_i for arbitrary values of the open loop gain A_0.

Figure 4.10(a) shows a simplified version of the bipolar weight memristive synapse circuit, where the memristors have been replaced with resistors, the programming switch has been removed, and the inputs are ideal current sources. The small signal model of the input stage of the post-synaptic neuron is also shown. Generally, each neuron will have multiple synapses connected to its input, labeled as *common node* (this is also the negative input of the post-synaptic neuron's opamp). Shown here is the synapse that connects the j^{th} neuron in the network to the i^{th} neuron. When the finite gain is considered, the value of s_i changes such that each synapse has an effective weight value of:

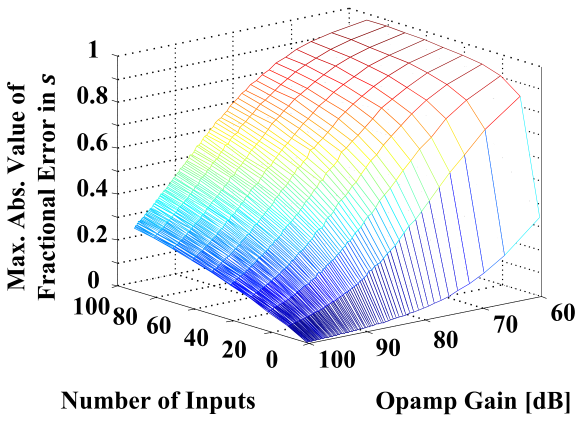
$$w_{i,j} = \left(2\frac{G_{m1i,j}}{G_{m1i,j} + G_{m2i,j}}\right)\left(\frac{1 + A_0 + \frac{G_{m2i,j} R_{in} i^*_{s_i}}{2 i_{x_j}}}{1 + A_0 + \sum_k \frac{G_{m1i,k} G_{m2i,k} R_{in}}{G_{m1i,k} + G_{m2i,k}}}\right) - 1, \tag{4.19}$$

where the index k has the same range as j, and $i^*_{s_i}$ is the sum of all of the individual synapse input currents. When the opamp gain A_0 is large, $w_{i,j}$ reduces to the original definition. However, when the gain is smaller, the accuracy of the synapse circuit is degraded. This is illustrated in Figure 4.10(b), where the maximum absolute value of the fractional error in s_i

[1]Note that here, s_i is normalized to a voltage instead of a current.



(a)



(b)

Figure 4.10: (a) Simplified model of the bipolar weight memristive synapse connected to the input of a post-synaptic neuron. (b) Number of neuron inputs and opamp gain vs. the maximum absolute value of the fractional error between the total synaptic output current and the ideal total synaptic output current.

is plotted versus the number of synapses connected to neuron i and the gain of the neuron's opamp. The fractional error is defined as $|(\bar{s}_i - s_i)/\bar{s}_i|$, where $\bar{s}_i$ is the expected ($A_0 = \infty$) sum of all synapse outputs divided by I_{max}. 1000 sets of random values for each synapse's conductance and input current were picked for each set of independent parameters (number of inputs and opamp gain). The value of s_i was determined using (4.18) and (4.19), while $\bar{s_i}$ was determined using (4.18) and (4.11). The maximum value plotted in Figure 4.10(b) is the maximum over all 1000 sets of random parameters. For this work, an opamp was designed with gain $A_0 > 100$ dB, which enables neurons to have ≈50 synaptic inputs while keeping the fractional error below $\approx 20\%$. The opamp design uses a high-gain folded cascode input stage and a common source output stage.

When an opamp is not needed (for virtual ground or otherwise), a single grounded resistor or memristor can be used for the neuron's input stage. A memristor may be a better choice because it will be significantly smaller and can be adjusted to change the shape of the neuronal response. In that case, the ideal input current from the synapses is summed across the resistor (or memristor) as

$$s_i = \sum_j w_{i,j} x_j I_{max} R_{in}. \tag{4.20}$$

The only difference between (4.18) and (4.20) is the sign. However, consider the case where the converging synapses are current mirror-based designs, as in Figure 4.11. If the current mirrors have short channel lengths, then their output will depend exponentially on v_{ds}, which is related to v_{s_i} in this case. However, the output current also depends linearly on v_{s_i} through Ohm's law: $i_{s_i} = v_{s_i}/R_{in}$. This leads to a transcendental equation in v_{s_i} that can be approximated as (see Appendix A)

$$s_i \approx \frac{2C}{I_{max}\left(-B + \sqrt{B^2 - 4AC}\right)}, \tag{4.21}$$

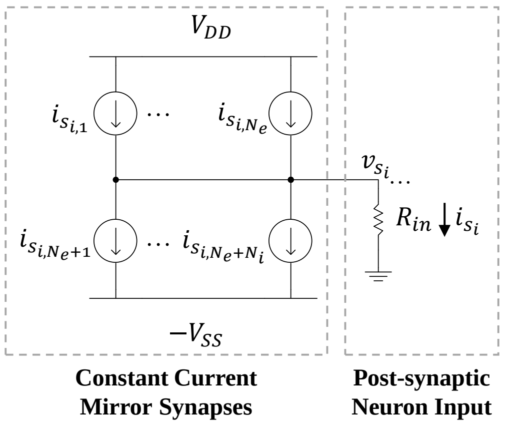
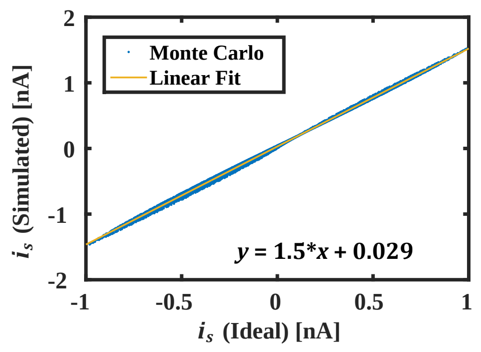


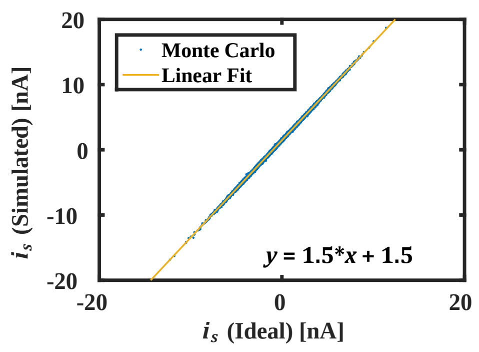
Figure 4.11: Constant current mirror synapses converging on a resistive input stage of a post-synaptic neuron.

where A, B, and C are constants that depend on the synaptic weights, pre-synaptic outputs, and channel lengths.

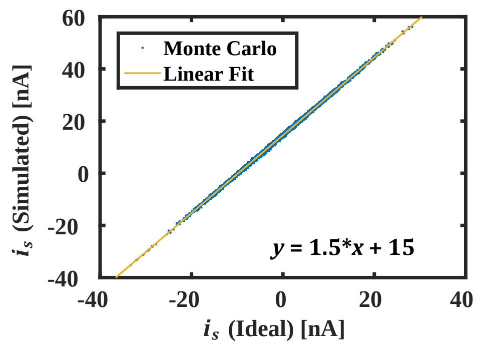
The effect of this non-linear input is illustrated in Figure 4.12, where the circuit in Figure 4.11 was simulated in HSPICE for 6 different cases. In the first 3 simulations (4.12(a)-4.12(c)) all transistor channel lengths are 45 nm, and the number of input synapses N is 2, 100, and 1000 for 4.12(a), 4.12(b), and 4.12(c), respectively. In each case 100000 Monte Carlo runs were simulated, varying the outputs of the pre-synaptic neurons (between 0 and 1). The magnitude of the weights was kept constant at 1. Similarly, Figures 4.12(a)-4.12(c) show simulation results for 2, 100, and 1000 inputs, but with the channel length set at 90 nm. Along with the Monte Carlo data are linear fits, with the equations shown in each plot. Ideally, the slope and intercept would be 1 and 0, respectively. However, it is observed that the slope is larger for smaller channel lengths and the intercept increases with and increasing number of inputs. This behavior is predicted by the model in (4.21) and is intuitive once one realizes that the effect of v_{ds} is stronger for smaller channel lengths (yielding the increasing slope) and also stronger for PMOS devices (yielding the increasing



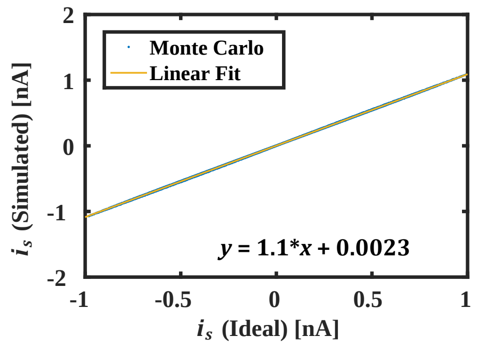
(a)



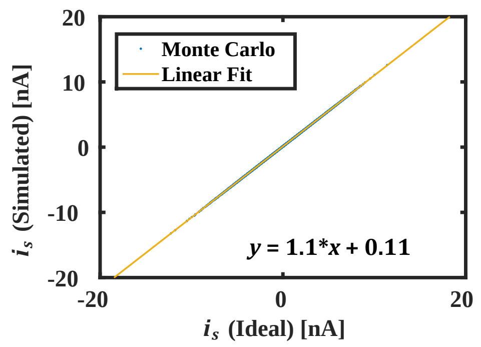
(b)



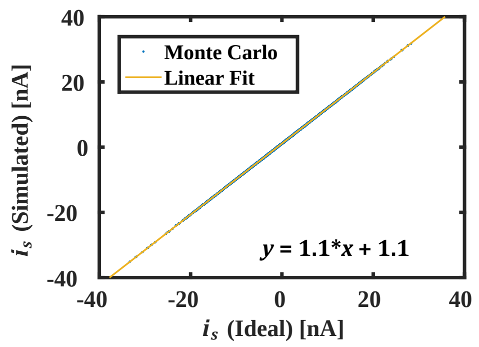
(c)



(d)



(e)



(f)

Figure 4.12: i_{s_i} vs. ideal i_{s_i} for a number N of constant current mirror synapses converging on a resistive input stage of a neuron. (a) $L = 45$ nm, $N = 2$. (b) $L = 45$ nm, $N = 100$. (c) $L = 45$ nm, $N = 1000$. (d) $L = 90$ nm, $N = 2$. (e) $L = 90$ nm, $N = 100$. (f) $L = 90$ nm, $N = 1000$.

intercept). This analysis also reveals that a fairly large fanin is possible without much non-linearity. However, it is important to realize that this work has not considered the layout of such circuits, which would be challenging in terms of interconnect routing.

4.2.2 Sigmoid and Hyperbolic Tangent Activation Functions

4.2.2.1 Basic Operation

The second stage of a neuron applies an activation function to the summation of its inputs. Sigmoid and hyperbolic tangent activation functions are most commonly used in artificial neural networks. Their saturating behavior closely approximates firing rates exhibited by biological neurons. More importantly, because they are smooth and have continuous derivatives, these activation functions can be used in gradient-based neural network training algorithms such as backpropagation.

The current-mode sigmoid/tanh neuron circuit is shown in Figure 4.13. The circuit is essentially an NMOS differential amplifier. It is biased with a sink current equal to I_{max}, which is mirrored in with a simple current mirror. The lengths of the transistors in the current mirror are $L = 180$ nm to get better output impedance and current matching. Since there are fewer neurons than synapses, we can afford to have larger transistors in the neurons, allowing us to simplify the models by removing v_{ds} dependence. It can be shown [94] that, if $x_i = x_{i1}^{+} = x_{i2}^{+}$, then

$$x_i = f_{\text{sig}}\left(s_i\right) = \frac{1}{1 + \exp\left(\frac{-\zeta_1 s_i I_{max} R_{in}}{n V_T}\right)}, \tag{4.22}$$

where R_{in} is from the input stage, and controls the slope of the sigmoid function at $s_i = 0$. Here, it is assumed that the input voltage v_{s_i} comes from an (inverting) opamp input stage. If the input stage is non-inverting (e.g. resistive), then the inputs to the differential pair would be switched. The positive output current is mirrored to another branch where it is

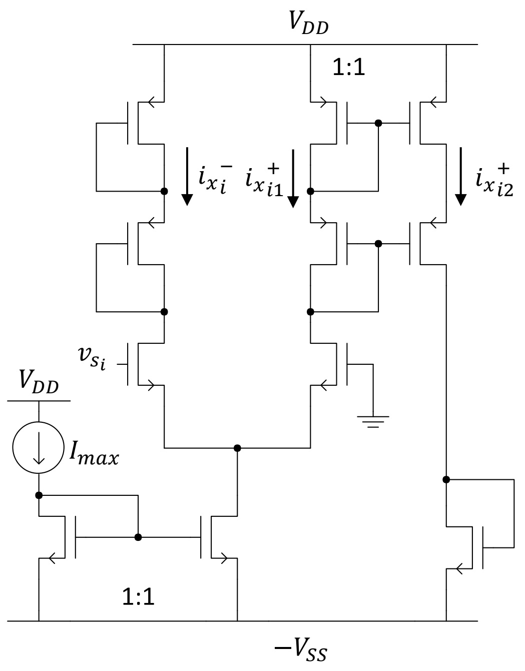


Figure 4.13: Current-mode sigmoid/tanh activation function circuit.

pulled down to $-V_{SS}$ though a diode-connected NMOS transistor. Therefore, the positive output current can be mirrored to a synapse through either a PMOS or an NMOS current mirror. This allows the neuron to connect to both excitatory and inhibitory constant current mirror synapses, as well the bipolar weight memristive synapse.

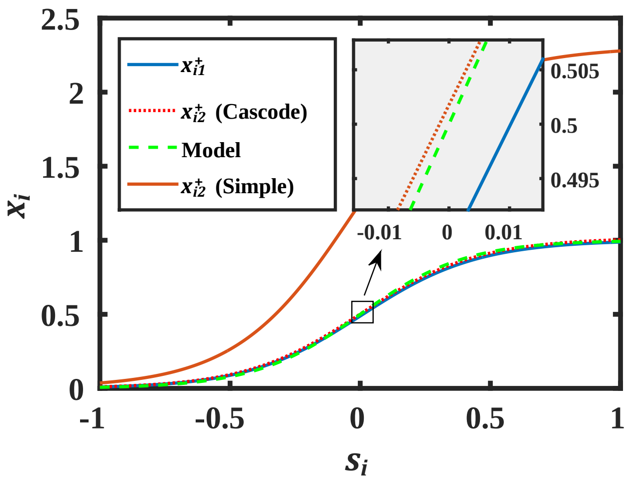


Figure 4.14: Transfer characteristics of the sigmoid activation function circuit.

The activation function's transfer characteristics are shown in Figure 4.14. The solid and dotted lines are results from HSPICE simulations for x_{i1}^+ and x_{i2}^+, respectively. The dashed line is from the model given in (4.22). The model gives high accuracy (see inset for a zoomed view at $s_i = 0$), with mean absolute errors being 0.013 and 0.009 for x_{i1}^+ and x_{i2}^+, respectively. Now, it is important to justify the use of a cascode current mirror in the activation function circuit, rather than a simple current mirror, which has only half of the area. The solid line in Figure 4.14 shows the HSPICE simulation result for x_{i2}^+, when a simple current mirror is used. The range of the output is $> 2\times$ the range when the cascode mirror is used. This is a direct result of a large v_{ds} value across the mirror's output transistor, which results in a large Π function for the mirror. Equivalently, the simple current mirror's output impedance is small, so the v_{ds} value of the output transistor can cause a large current mismatch. Since the neuron's power consumption will be related to the output currents, it is important to reduce x_{i2}^+. The current matching can be improved in several ways, such as increasing the channel lengths of the transistors in the simple current mirror, or adding voltage dividers to the x_{i2}^+ branch (i.e. reducing v_{ds} across the output transistor. However, it was found that the most area-efficient way to improve matching is to use a cascode mirror, which causes the output impedance to be squared (i.e. if the simple current mirror's output impedance is r_o, then the cascode mirror's output impedance will be r_o^2.

The circuit in Figure 4.13 can also be used to implement a hyperbolic tangent activation function. If $x_i = x_i^+ - x_i^-$, then

$$x_i = f_{\tanh}(s_i) = \tanh\left(\frac{\zeta_1 s_i I_{max} R_{in}}{nV_T}\right). \tag{4.23}$$

Notice that the two current components would have to be subtracted and then applied to the input of a synapse circuit. This work mostly focuses on the sigmoid, rather than the tanh activation function.

The sigmoid neuron used in this work has several advantages over other designs. For example, voltage-mode designs [95] have a limited input/output voltage range and have more complex designs. Current-mode neurons that operate in superthreshold [96] have increased power consumption and do not exhibit a true sigmoid activation. Digital designs, such as the one in [97] rely on piecewise linear approximations, yielding large area overhead.

4.2.2.2 Area and Power Consumption

The sigmoid activation function circuit is composed of 10 transistors (the input transistor for the bias current mirror is shared):

$$A_{sn} \approx A_{sn,bias} + 2A_{sn,dp} + 6A_{sn,pmirror} + A_{sn,nmirror}, \tag{4.24}$$

where $A_{sn,bias}$ is the area of the NMOS bias transistor, $A_{sn,dp}$ is the area of the differential pair transistors, $A_{sn,pmirror}$ is the area of the PMOS transistors in the cascode configuration, and $A_{sn,nmirror}$ is the area of the NMOS transistor for mirroring $i_{x_{i2}^+}$. Note that $A_{sn,pmirror}$ and $A_{sn,nmirror}$ will depend on the minimum sizing requirements for the outgoing synapses.

The activation function's power consumption is estimated as

$$P_{sn} \approx 2V_{DD}\left(i_{x_i^-} + i_{x_{i1}^+} + \eta_i i_{x_{i2}^+}\right), \tag{4.25}$$

where η_i is the neuron's activity factor. Nominally, $i_{x_i^-} + i_{x_{i1}^+} = I_{max}$. However, the sum may be larger or smaller due to process variations.

4.2.2.3 Process Variations

Variations in V_{th0} and R_{in} have four primary effects on the behavior of the sigmoid activation function. Figure 4.15 shows the results of 1000 Monte Carlo simulations while varying V_{th0}. Variations in the current mirror for I_{max} increase or decrease the maximum value of x_{i1}^+. Then, variations in the cascode current mirror cause x_{i2}^+ to vary from x_{i1}^+. Variations in the differential pair creates an offset current, effectively shifting the activation function left or right. Finally, variations in R_{in} (normally-distributed with 30% variation) cause the sigmoid slope to increase or decrease. The overall effect can be written as

$$x_{i1}^+ = \hat{f}_{sig_i}\left(s_i\right) = x_{max_i} f_{sig}\left(\left(1 + \frac{\Delta R_{in_i}}{R_{in}}\right) s_i + \Delta s_i\right). \tag{4.26}$$

The distributions of the maximum output x_{max_i} and its mirrored value x_{i2}^+ are modeled in the same way as the constant current mirror synapses. The offset Δs_i can be modeled as a Gaussian distribution with mean s_i and variance $\sigma^2\left(\Delta V_{th0}\right) / \left(R_{in} + \Delta R_{in_i}^2\right)$.

4.2.3 Periodic Activation Functions

4.2.3.1 Basic Operation

The sigmoid and tanh activation functions are monotonic and, as a result, can only implement one decision boundary in their input space. Non-monotonic activation functions, on the other hand, can implement several decision boundaries. Networks that non-monotonic

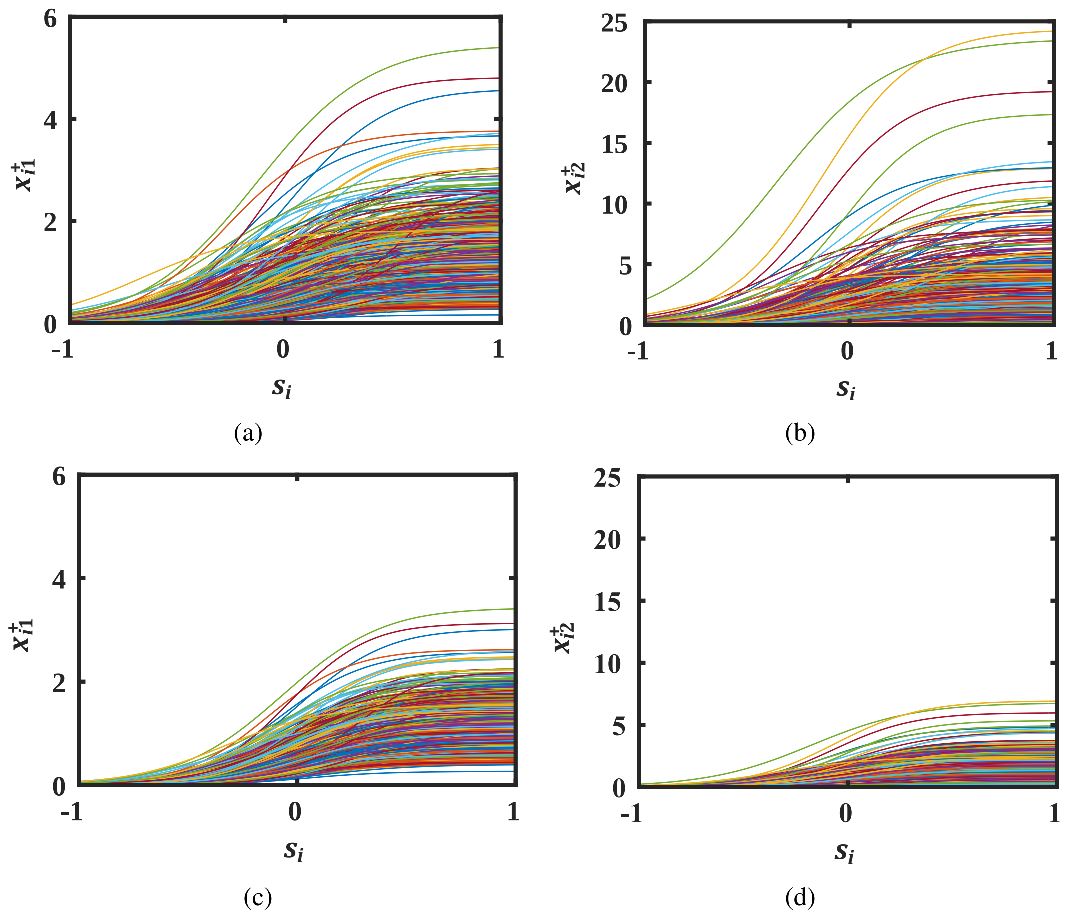


Figure 4.15: Monte Carlo simulations of the sigmoid neuron activation function with variation in V_{th0}. (a) and (b) have minimum sizing, except for the I_{max} current mirror, which has $L = 180$ nm. (c) and (d) have $W = 90$ nm and $L = 90$ nm, except for the I_{max} current mirror, which has $L = 180$ nm.

activation functions can learn more complex tasks with fewer neurons. This translates to lower area and power overhead. For example, in [98, 99], Soltiz et al., showed that neural networks employing digital square wave activation functions yield energy-delay products (EDPs) which are orders of magnitude less than other designs. This work introduces an analog counterpart to Soltiz's neuron design.

The activation function design is based on CMOS folding amplifiers, which are commonly used in high-speed analog-to-digital converters [100]. Folding amplifiers produce voltage outputs that are periodic in their input voltages. That functionality is exploited to

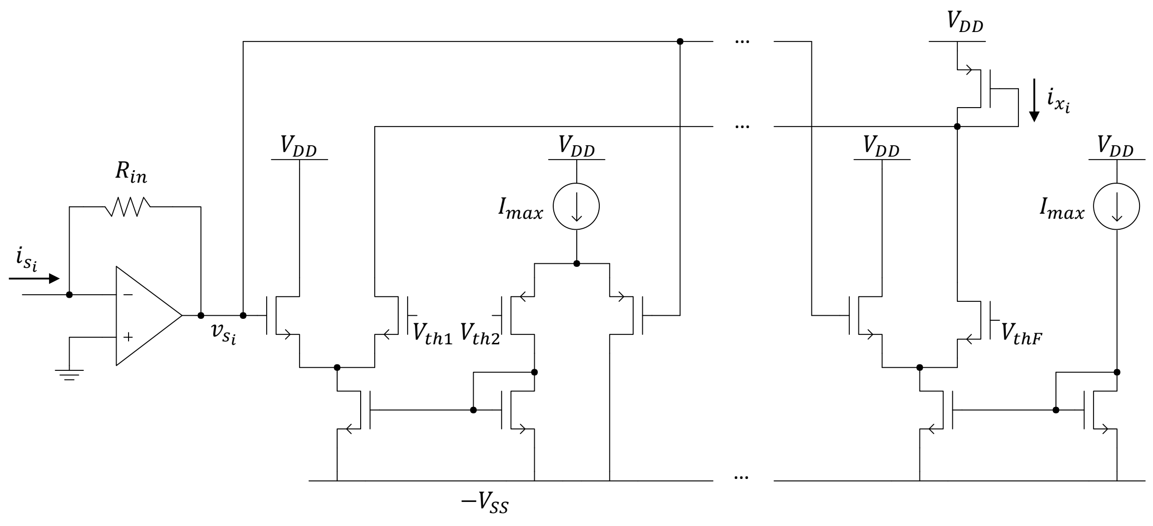


Figure 4.16: Folding amplifier activation function with an opamp input stage.

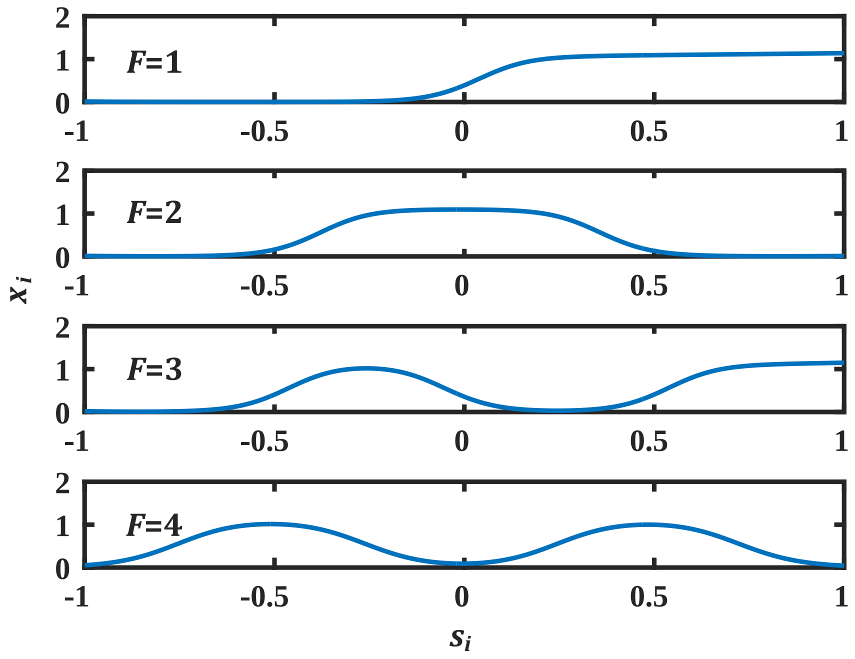


Figure 4.17: Folding amplifier activation function with different fold factors (F).

design an analog neuron with a periodic activation function (Figure 4.16). The folding factor F is the number of times that the amplifier's output makes a low-to-high or high-to-low transition. The folding amplifier's operation is straightforward. For each fold, there is an NMOS differential pair, biased by I_{max}. One differential input is connected to v_{s_i}, while the other is connected to a threshold voltage V_{thk}. When $v_{s_i} \ll V_{thk}$, all of the bias current flows through the transistor connected to the threshold voltage. When $v_{s_i} = V_{thk}$, an equal amount of current flows through both branches. Finally, when $v_{s_i} \gg V_{thk}$, all of the bias current flows through the transistor connected to the input voltage. Therefore, v_{s_i} can be used to modulate the current flowing through the output node. Figure 4.17 shows the voltage transfer characteristics of four different folding amplifiers with F = 1, 2, 3, and 4 from top to bottom. The first plot is similar to the commonly-used sigmoid transfer function. The second one, with $F = 2$ is similar to a radial basis function. The shapes of the functions can be adjusted by changing the threshold voltages and gain of the circuit. However, the exploration of that parameter space is saved for future work. The third and fourth plots are not commonly used in neural networks, but allow each neuron to learn a broader class of non-linearly separable functions. In fact, a single neuron with folding factor F can learn functions with F decision boundaries.

4.2.3.2 Area, Power Consumption, and Process Variations

The area of the folding amplifier's periodic activation function can be estimated as

$$A_{pn} \approx F\left(A_{pn,bias} + 2A_{min}\right) + \left\lfloor \frac{F}{2} \right\rfloor A_{pn,bias}, \tag{4.27}$$

where $A_{pn,bias}$ is the area of the current mirror and current source transistors, which have longer channels to reduce the effect of v_{ds}. The power consumption of the periodic activation function circuit is

$$P_{pn}^{avg} \approx \left(F + 1\right) I_{max} V_{DD}. \tag{4.28}$$

The effect of process variations on the folding amplifier activation function has not been modeled in this work. However, the development of such a model will follow a similar derivation as for the sigmoid/tanh activation function circuit. In particular, variations will affect the maximum output and the position(s) of the function's transistion(s) from 0 to 1 or 1 to 0.

4.2.4 Additional Activation Functions

Additional activation function circuits used in this work include linear, rectified linear and threshold functions, which are described by

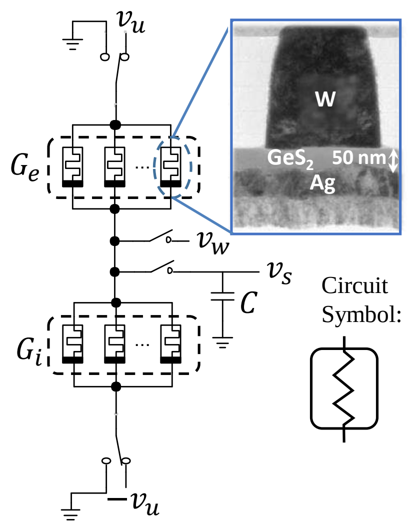
$$f_{lin}\left(s_i\right) = Bs_i, \tag{4.29}$$

$$f_{rlin}\left(s_i\right) = H\left(s_i\right)Bs_i, \tag{4.30}$$

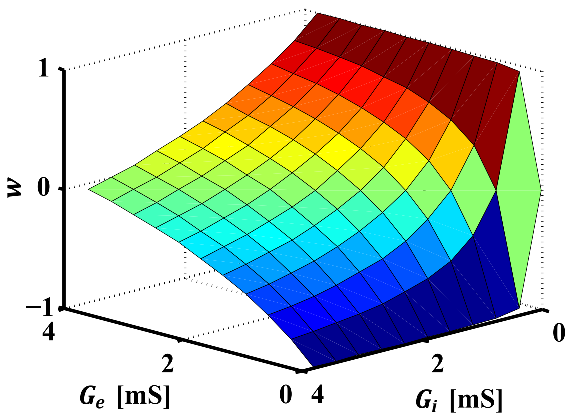
and

$$f_{thresh}\left(s_i\right) = H\left(s_i - \theta\right), \tag{4.31}$$

where B is a slope factor (sometimes called a boost factor), and θ is a threshold value. The linear activation functions are implemented using diode-connected MOSFETs and current mirror, while the threshold activation function is implemented with a current comparator [101].



(a)



(b)

Figure 4.18: (a) Proposed synapse circuit consisting of excitatory and inhibitory groups of Ag/GeS$_2$ CBRAM devices. (b) Effective synaptic weight vs. individual conductances (G_e, G_i).

4.3 Voltage-Mode CBRAM Synapse and Neuron Circuits

A large number of the memristive devices reported in the literature exhibit low-resolution discrete switching operation. Moreover, progress in memristor device and fabrication research is primarily being driven by a push for new non-volatile memory technology. Consequently, the majority of near-term commercial memristor processes will be optimized for discrete, likely bi-stable, behavior. This work exploits the stochastic behavior of bi-stable CBRAM devices, to achieve quasi-continuous behavior. Previous work [75, 102] has explored this idea primarily at the device level and proposed system-level models that embed the device characteristics. However, there has been little work that comprehensively studies the implications of experimental device behavior on circuit and system-level design.

The CBRAM-based synapse circuit multiplies an input value v_u by a weight value w to produce an output v_s. The design (Figure 4.18(a)) consists of two sets of k CBRAM devices (Ag/GeS$_2$/W stack, adopted from [75]) connected in parallel. The first set G_e forms the positive (excitatory) input, which allows the capacitor C to charge towards v_u. The second set G_i forms the negative (inhibitory) input, that allows C to charge towards $-v_u$. After discharging the capacitor, the positive and negative inputs are applied for a period of time t (we have used t=100 ns). Then, the inputs are disconnected. The final synaptic output voltage v_s, is given by

$$v_s = v_u \left(\frac{G_e - G_i}{G_e + G_i} \right) \left[1 - \exp \left(-t \frac{G_e + G_i}{C} \right) \right] = v_u w. \tag{4.32}$$

Figure 4.18(b) shows the range of weights w that can be achieved in an individual synapse w.r.t. CBRAM device conductances. Unlike previous works [17, 103], our synapse can attain both positive and negative values for w which is important for fitting different types of data. The circuit's inputs can also be switched from $\pm v_u$ to ground for programming via a write voltage v_w. Each CBRAM device switches probabilistically (and independently) with the applied write voltage. This fact is critical for the synapse to achieve multiple weight states. If all of the devices switched simultaneously with 100% probability, then each synapse would only have 2 weight states, severely limiting the precision of the proposed system. In fact, in order to build a similar system with devices that switch deterministically, we would need individual control of each CBRAM device, which would significantly increase the design complexity and area overhead.

A charge sharing-based neuron can be implemented by connecting outputs from a number of CBRAM synapses N together by switches. When the switches are closed, a common node is formed with a voltage given by:

$$v_{\hat{y}} = \frac{1}{1+N} \sum_i v_{s_i} \tag{4.33}$$

This is accomplished by sharing the charge on all of the synapses' output nodes using the switches (transmission gates) in the neuron circuit. Although this type of charge sharing scheme has been used elsewhere for neuron design (cf. [104]), we note an important property that has been previously overlooked. A common issue in regression problems is that the training process will find a model that fits the training data very well, but the model will fit unseen data from the same distribution very poorly. This is called overfitting. One method that is often used to combat overfitting is regularization, where the weights in the models are kept small. Generally, the more weights there are, the more flexible the model will be, leading to a higher probability of overfitting. Therefore, in our system, it is desirable to have smaller weight values when the number of inputs to the system is larger. It can be seen from (4.33) that this is a natural property of our system. That is, the *effective* weight of each synapse is scaled by $1/(1+N)$, leading to smaller weights for larger systems.

4.4 Summary

This chapter presented designs and models for synapse and neuron circuits that can be integrated into NMSs. The novel contributions discussed in this chapter are

- Design and modeling of current-mode synapses that leverage process variations for random weight distributions

- Design and modeling of a current-mode bipolar weight memristive synapse that has reduced area and power consumption
- Design and modeling of neuron circuits with sigmoid/tanh, periodic, linear, rectified linear, and threshold activation functions
- Design and modeling of a voltage-mode regression circuit that leverages the stochastic switching nature of CBRAM synapses

The synapse and neuron circuits discussed in this chapter provide essential building blocks of an NMS. The next chapter discusses methods for training an NMS by modifying synaptic weight values.

Chapter 5

Synaptic Plasticity Circuits

Synaptic plasticity facilitates the modulation of connection strength between different neurons in an NMS. It is the primary mechanism of learning and adaptation employed in these systems. Within an NMS, it is critical to have efficient circuits that facilitate synaptic plasticity. Synapses are modified via a particular training algorithm which can be unsupervised, supervised, or semi-supervised. However, there are some common characteristics among each of these. One commonality is that each algorithm typically involves the product of two quantities, such as a pre-synaptic neuron's output and an error value. Multiplication of digital values is easily achieved using an AND gate, but analog multiplication is more costly, requiring at least a Gilbert Multiplier. This chapter discusses novel training algorithms to reduce the cost of training circuits. First, a stochastic least-mean-squares algorithm is developed for both online and batch mode training. Then, another approach to is developed, where multiplication on the unit square is approximated by a minimum function

5.1 Online SLMS Algorithm

5.1.1 Overview

Consider the simple linear regression problem, where m datapoints $(\mathbf{u}^{(p)}, y^{(p)}) \in \mathbb{R}^{N+1}$ are to be modeled by a straight line fit:

$$\hat{y}^{(p)} = h_{\mathbf{w}}(\mathbf{u}^{(p)}) = w_0 + w_1 u_1^{(p)} + w_2 u_2^{(p)} + \cdots + w_N u_{N-1}^{(p)}. \tag{5.1}$$

where $\mathbf{u}^{(p)} \in \mathbb{R}^N$ are the independent variables, $\hat{y}^{(p)}$ is the model estimate of $y^{(p)}$, and $\mathbf{w} = [w_0, w_1, \cdots, w_N]$ is a parameter vector of the unknown line coefficients. Note that we always assume that $u_0^{(p)}$ is a bias input and therefore has a value of 1. A common strategy to solve this problem is to define a cost, or error function as

$$J(\mathbf{w}) = \frac{1}{2k} \sum_{p=1}^{m} \left(h_{\mathbf{w}}(\mathbf{u}^{(p)}) - y^{(p)} \right)^2 \tag{5.2}$$

and minimize it by adjusting w_j as

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(\mathbf{w}) \tag{5.3}$$

where α is a constant called the learning rate. This leads to the common batch gradient descent or least-mean-squares (LMS) algorithm:

$$w_j := w_j + \frac{\alpha}{m} \sum_{p=1}^{k} \left(y^{(p)} - \hat{y}^{(p)} \right) u_j^{(p)}, \tag{5.4}$$

where each w_j is updated simultaneously. The summation in (5.4) presents a challenge in hardware, especially if m is large (which it usually is). To circumvent this, there is a simpler version that estimates the partial derivative in (5.3) as [105]

$$\frac{\hat{\partial}}{\partial w_j} J(\mathbf{w}) = \left(y^{(p)} - \hat{y}^{(p)}\right) u_j^{(p)}. \tag{5.5}$$

Finally, plugging this estimate into (5.3) gives the online version of the LMS algorithm:

$$w_j^{(p)} := w_j^{(p)} + \alpha \left(y^{(p)} - \hat{y}^{(p)}\right) u_j^{(p)}. \tag{5.6}$$

The weight update rule in (5.6), although fairly simple, involves multiplications, which are costly to implement in analog hardware. Ideally, we can eliminate them.

The approach used in this work is to convert all of the analog variables in (5.6) to random variables with Bernoulli distributions. Concretely, for each variable z, a random variable Z is created such that

$$Z \sim \mathrm{B}(1, z) \tag{5.7}$$

where B is the Binomial distribution. This leads to a new weight update equation:

$$w_j^{(p)} := w_j^{(p)} + \alpha \left(Y^{(p)} - \hat{Y}^{(p)}\right) U_j^{(p)} \tag{5.8}$$

where each uppercase variable is defined the same way as in (5.7). Finally, we notice that using this weight update rule may cause the sign of the weight change to be negative when it should be positive (or vice versa). Therefore, we make a slight modification, leading to the proposed algorithm for linear regression:

$$w_j^{(p)} := w_j^{(p)} + \alpha \text{sign}\left(y^{(p)} - \hat{y}^{(p)}\right) |Y^{(p)} - \hat{Y}^{(p)}| U_j^{(p)}. \tag{5.9}$$

At first glance (5.9) looks complex, but we will show in the next section that it can be implemented in hardware using only comparators and digital logic gates. We call this the stochastic least-mean-squares (SLMS) algorithm. If we treat the change in a weight value $\Delta w_j^{(p)}$ as a random variable, then it is easy to show that its expected value is

$$\begin{aligned} E\left[\Delta w_j^{(p)}\right] = \ & \alpha \text{sign}\left(y^{(p)} - \hat{y}^{(p)}\right) \\ & \times \left(y^{(p)} + \hat{y}^{(p)} - 2y^{(p)}\hat{y}^{(p)}\right) u_j^{(p)}. \end{aligned} \tag{5.10}$$

The second term in parenthesis $y^{(p)} + \hat{y}^{(p)} - 2y^{(p)}\hat{y}^{(p)}$ behaves similar to the absolute value function $|y^{(p)} - \hat{y}^{(p)}|$ except when $y^{(p)}$ and $\hat{y}^{(p)}$ are approximately equal and midway through their range (i.e. 0.5). This gives us some mathematical justification for using this algorithm.

In the discussion above, we developed a learning algorithm for NMSs. Specifically, we have shown that the algorithm works well for linear regression problems. It is easy to show that the SLMS algorithm will also work well for logistic regression, where several datapoints are to be classified as belonging to one of the two groups, or classes. In this case

$$\hat{y}^{(p)} = h_{\mathbf{w}}(\mathbf{u}^{(p)}) = \frac{1}{1 + \exp\left(B\mathbf{w}\mathbf{u}^{(p)}\right)}, \tag{5.11}$$

where B is a constant that scales the sigmoid's slope. By defining the cost function as

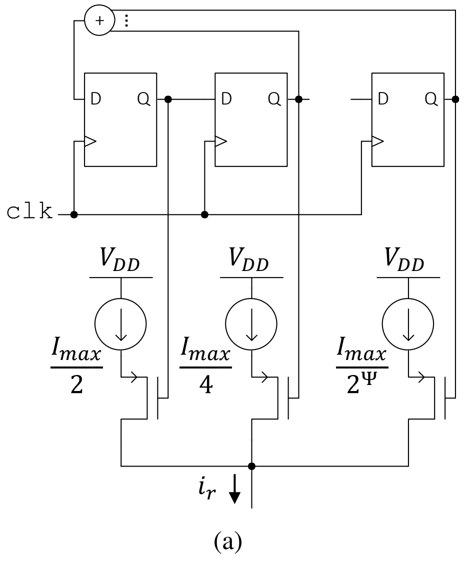
$$J(\mathbf{w}, y^{(p)}) \begin{cases} -\ln(h_{\mathbf{w}}(\mathbf{u}^{(p)})) & y^{(p)} = 1 \\ -\ln(1 - h_{\mathbf{w}}(\mathbf{u}^{(p)})) & y^{(p)} = 0 \end{cases} \tag{5.12}$$

we get an online weight update equation identical to (5.6), so the stochastic implementation can be applied to classification problems as well.

5.1.2 Hardware Implementation

This section discusses a hardware implementations of the proposed SLMS algorithm. For comparison, we also provide a hardware implementation of the LMS algorithm. We estimate the area of both implementations as a function of the number of the number of independent variables N. Although many hardware memristive synapse (weight) designs have been proposed [33–36, 38, 39, 89, 106], we make a couple of simplifying generalizations: 1.) The synapse has bipolar weights which can have values between -1 and +1. 2.) The synapse has two terminals that are used for applying positive and negative write voltages (v_{w+} and v_{w-} to change the weight value. 3.) The magnitude of $v_{w+} - v_{w-}$ must exceed a threshold before any weight change will be effected. 4.) The synapse's weight value will change proportionally to the flux of the write voltage applied to those two terminals (provided condition 3 is met). In other words, $\Delta w_j \propto (v_{w+} - v_{w-})t_\alpha$, where t_α is the amount of time that we apply the write voltage.

The first step in implementing the proposed SLMS algorithm, shown in (5.9), is to convert the expected NMS output $y^{(p)}$, the actual output $\hat{y}^{(p)}$, and each input $u_j^{(p)}$ to Bernoulli-distributed random variables, as described in the last section. A straight-forward method



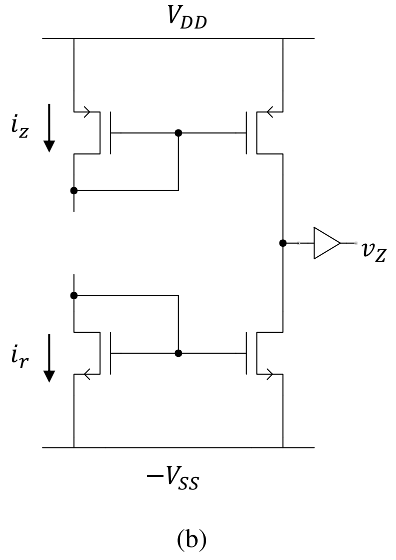


Figure 5.1: (a) Random current generator circuit. (b) Random current comparator (RCC) for converting an analog current i_z to a Bernoulli-distributed digital voltage v_Z.

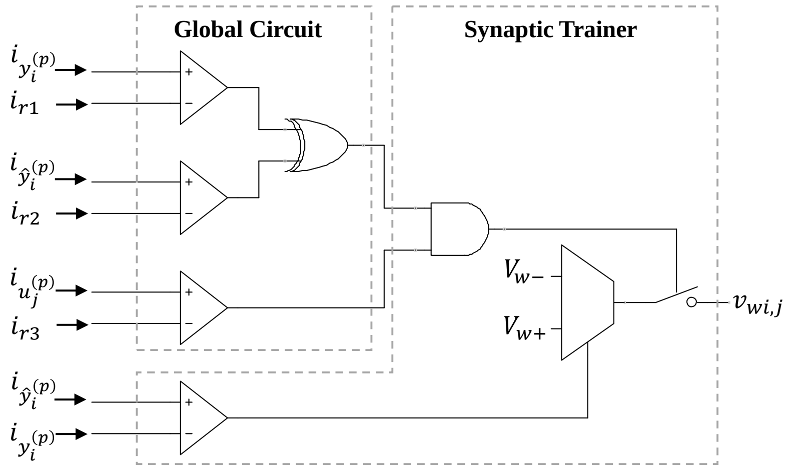


Figure 5.2: Hardware implementation of the online SLMS algorithm.

is compare each value z (where z can be any of the analog variables) to a uniformly-distributed random value r. If $z > r$, then $Z = 1$. Otherwise, $Z = 0$. Therefore, we need a method for generating random numbers that take on real values with the same change as z's. The proposed circuit is shown in Figure 5.1(a). A linear feedback shift register (LFSR) is used to generate pseudorandom digital values. Then, a simple binary-weighted digital-to-analog converter. The output current i_r will range from 0 to $I_{max} \sum_{i=1}^{\Psi-1} 1/2^i$. Here, Ψ is the number of flip flops in the LFSR. The upper bound on the index i is $\Psi - 1$ rather than Ψ because the LFSR output will never be all zeros. Futhermore, the resolution i_r will be $I_{max}/2^{\Psi}$. As an example, when Ψ=4, i_r will range from 0 to (7/8)I_{max} with a resolution of I_{max}/16. Finally, I_{max} should be set to

$$I_{max} = \frac{\max(z)}{\sum_{i=1}^{\Psi-1} 1/2^i}, \tag{5.13}$$

where z is the maximum value that an analog variable in the NMS can take on. The random current i_r is compared to a random variable i_z (we assume that we have a current-mode NMS implementation) using the current comparator shown in Figure 5.1(b). Two current mirrors drive a common node, which is connected to the input of a digital buffer. The output of the buffer is a Bernoulli-distributed random variable representation of z.

Now, it can be shown that combining the circuits in Figure 5.1 with some simple logic gates allows a direct implementation of the proposed SLMS algorithm. The circuit architecture is shown in Figure 5.2. It is split into two parts: a global circuit and synaptic trainer. In the global circuit, each independent current source is a random current generator like the one shown in Figure 5.1(a). It is important that the random seed in the LFSRs of each current source is different, so their outputs will be statistically independent. The global trainer takes current representations of the expected and actual outputs and converts them into random variables, as described above. We use an XOR gate to compute the absolute value of $Y^{(p)} - \hat{Y}^{(p)}$. Finally, this value is multiplied by an enable value (en) using an AND gate. The enable signal is held high for t_{α}. This is one of two outputs of the global circuit. The other output is the sign of the error $y^{(p)} - \hat{y}^{(p)}$. Again, we use the current comparator design in Figure 5.1(b), with a current representation of the expected output as the positive input and a current representation of the actual output as the negative input. The output will be a digital '1' if $y^{(p)} > \hat{y}^{(p)}$ and '0' otherwise.

The two outputs of the global circuit are used by each synaptic trainer to modify the synaptic weight values. There is one synaptic trainer per input (N). The input value $u_j^{(p)}$ is converted to a random variable just as the expected and actual outputs. The absolute value output of the global circuit is multiplied with $U_j^{(p)}$ using an AND gate. This value and the sign of the error from the global circuit are used to generate the synaptic write voltage.

We consider a straightforward implementation of the LMS algorithm based on a four-quadrant Gilbert multiplier with voltage inputs and current outputs [107, 108]. The biggest challenge in implementing the LMS algorithm is caused by the threshold assumption that we made earlier. That is, each synapse has a threshold voltage that must be exceeded in order to change its weight value. This is illustrated in Figure 5.3. The left plot shows the product $(y_i^{(p)} - \hat{y}_i^{(p)})u_j^{(p)}$. The synapse's threshold voltage is also shown in the vertical axis. When the product is positive, the result needs to be shifted up to the threshold voltage, and when it's negative it needs to be shifted down. The result is shown on the right. We can easily achieve this in hardware using the sign of the error and level shifters.

The circuit is shown in Figure 5.4. A global circuit computes the sign of the error, the same way as in the SLMS implementation. In the synaptic trainer, the inputs to the Gilbert multiplier are level shifted (LS block) to satisfy the multiplier's input range. If the product is negative (the multiplier's output is a differential current), then it is shifted down by i_s. Otherwise, it is shifted up by i_s. The value of i_s depends on the pull-down resistors R_{pd} which are used to convert the multiplier's shifted output to a differential voltage. Output stages (common source amplifiers in this case) are used to provide low output impedance in order to drive the synapse load. Finally, as in the case of the SLMS implementation, we add an enable signal which is held high for t_α.

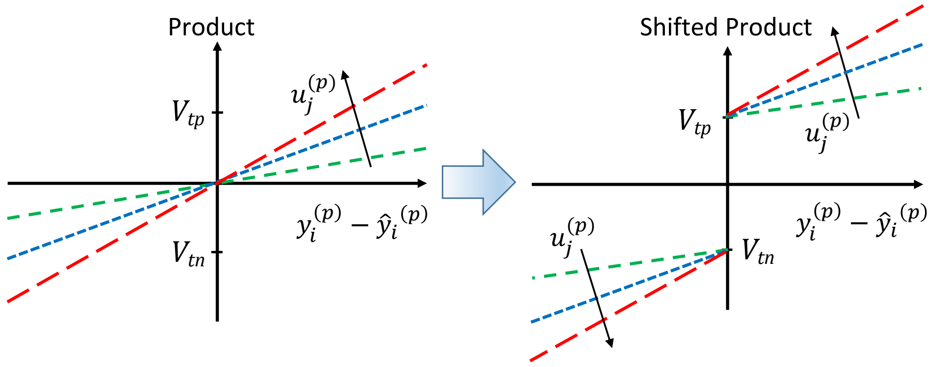


Figure 5.3: Illustration of the transformation that must be applied to the output of the Gilbert multiplier to implement the LMS algorithm: (Left) Unshifted multiplier output and (Right) output after applying the shift.

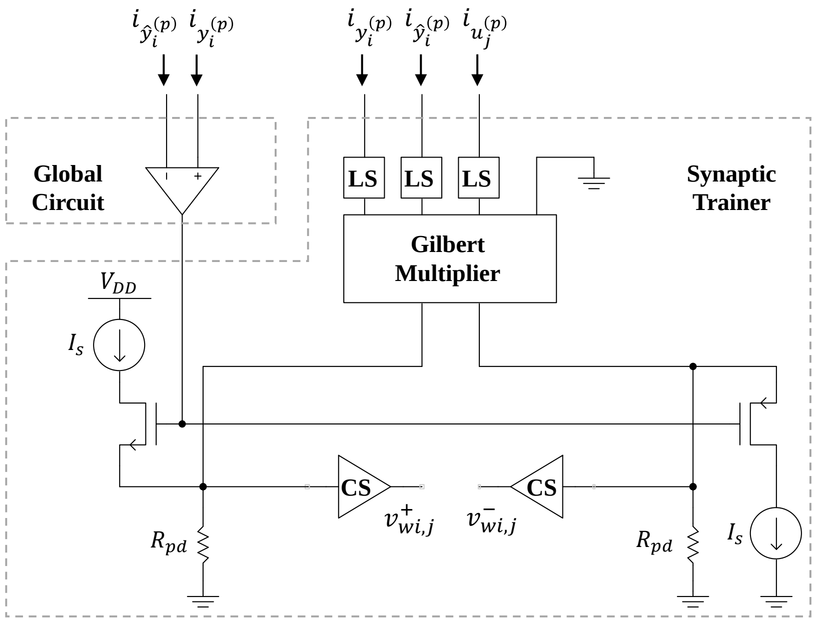


Figure 5.4: Hardware implementation of the online LMS algorithm.

Let us now derive an expressions to estimate the area costs of the SLMS and LMS hardware implementations. We will assume that the total area of the implementation is roughly equal to the sum of the areas of the components, and the areas of the components are equal to the areas of their constituent transistors. It is important to note that this is a simplified estimation. Further optimizations are possible at layout level. In addition, transistors in digital circuits (e.g. inverters) have minimum sizing (and area) and transistors in analog circuits have some multiple κ of the minimum sizing. Analog circuits are typically much larger than digital circuits for various reasons. For example, the length of transistors in analog circuits is often increased to improve the output impedance of a circuit, and the width is usually increased to carry a specified current. Furthermore, the total transistor area is often increased to improve matching properties. Table 5.1.2 shows a breakdown of the area of the global circuit and synaptic trainer for both the SLMS and LMS algorithms. Each entry is a multiple of the minimum transistor area. As an example, the current comparator circuit has 2 current mirrors, each with 2 analog transistors, for a total of 4 analaog transistors. It also has 4 transistors with minimum sizing for the buffer, yielding an area of 4κ+4. There are 3 of these circuits in the SLMS's global circuit, one in it's synaptic trainer, and one in the LMS's global circuit.

We notice that the cost for the SLMS's global circuit is much higher than that of the LMS circuit and vice versa for the SLMS's synaptic trainer. However, there is one synaptic trainer for every input, so the area cost for the SLMS and LMS implementations are

$$A_{SLMS} = 4\kappa\Psi + 22\Psi + 12\kappa + 24 + 24N + 4N\kappa \tag{5.14}$$

and

$$A_{LMS} = 4\kappa + 4 + 28N\kappa, \tag{5.15}$$

Table 5.1: Comparison of online SLMS and LMS area.

	SLMS		LMS	
Component	**Global Circuit**	**Synaptic Trainer**	**Global Circuit**	**Synaptic Trainer**
Digital				
AND2	6	6	-	-
XOR	$6\Psi + 6$	-	-	-
DFF	16Ψ	-	-	-
Analog				
Switch	$\kappa\Psi$	3κ	-	4κ
Current Comparator	$3(4\kappa+4)$	$4\kappa+4$	$4\kappa+4$	-
Current Source/Sink	$3\kappa\Psi$	-	-	2κ
Resistor	-	-	-	2κ
Multiplier	-	-	-	10κ
Level Shift	-	-	-	6κ
CS Amp.	-	-	-	4κ
Total	$4\kappa\Psi+22\Psi$ $+12\kappa+24$	$7\kappa+24$	$4\kappa+4$	28κ

respectively. Based on these expressions, we expect $A_{SLMS} > A_{LMS}$ for small N and $A_{SLMS} < A_{LMS}$ for larger N. In the next section, we will evaluate the area and performance of each implementation for two different applications.

5.1.3 Algorithm Performance

For the linear regression problem we took 11 linearly-spaced points lying on random lines $y = w_0 + w_1 u_1$ (random values for w_0 and w_1). We added Gaussian noise to each point and used the resulting points as the data to be regressed. The results are shown in Figure 5.5. Figure 5.5(a) shows the mean squared error (MSE) versus training epoch for a single run of both the SLMS and LMS algorithms. In this case, α=0.001. The SLMS algorithm takes longer to converge. This is an expected result since it will often not adjust weight values in cases where the LMS algorithm would. Figure 5.5(b) shows random line data and the linear fits found via the SLMS and LMS algorithms. In Figure 5.6 we show the mean

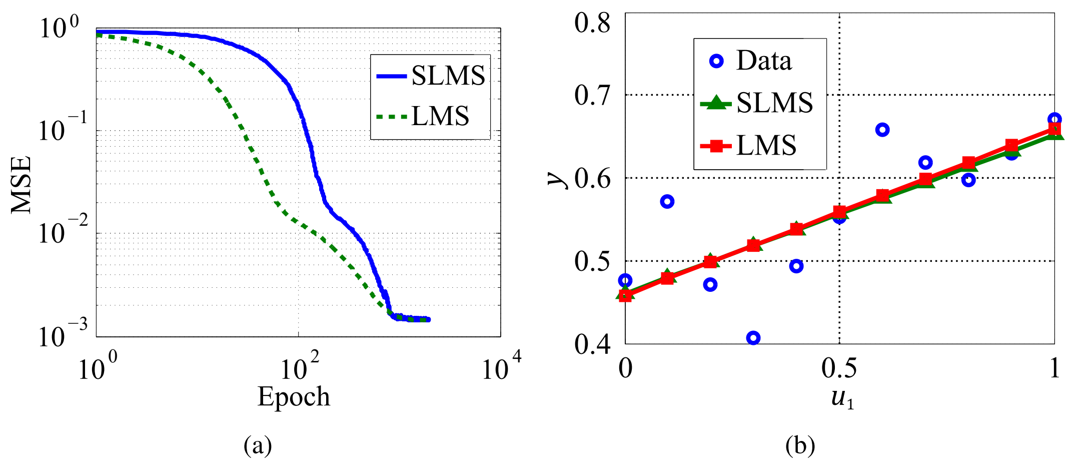


Figure 5.5: Performance of the proposed algorithm on a linear regression problem: (a) MSE vs. training epoch for the LMS and SLMS algorithms. (b) Datapoints from a straight line with random Gaussian noise added and the linear fits provided by the LMS and SLMS algorithms.

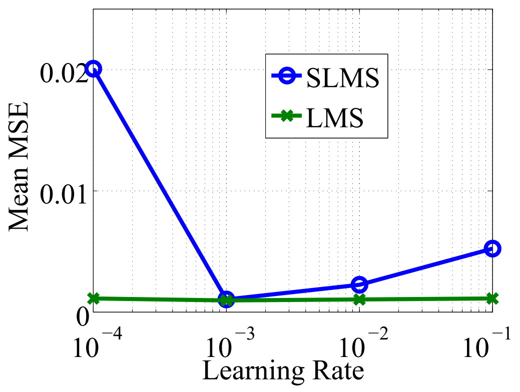


Figure 5.6: Mean MSE versus α (learning rate) over 10 runs.

MSE value over 10 runs of each algorithm for 4 different learning rates. As expected, the LMS algorithm is relatively insensitive to learning rate. The SLMS algorithm, however, will generally have higher MSE with a larger learning rate. This is because the learning rate defines the resolution of the weight change for the SLMS algorithm. If the resolution is low (large learning rate), then the weights cannot be fine-tuned enough to fit the data well. Interestingly, however, at very small values of α, the MSE is also large. We believe this is tied to the slower convergence of the SLMS algorithm.

Assuming Ψ=10, κ=10, and with N=2, we have area costs of A_{SLMS}=892 and A_{LMS}=604. Therefore, for such a small number of inputs, the area cost of the proposed algorithm is much larger ($\approx$1.5$\times$) than the LMS implementation.

5.2 Batch SLMS Algorithm

Gradient descent approaches for training neuromorphic systems can be expensive in terms of area overhead and design complexity, especially in the case of analog designs. We have addressed this problem in [109] by designing a stochastic version of the online least-mean-squares (LMS) training algorithm [110], which was able to achieve $\approx$3.5$\times$ area reduction and similar accuracy when compared to a conventional LMS implementation. The SLMS algorithm converts the signals used for training from analog values to stochastic values from a Bernoulli distribution. In this work, we have designed a batch-mode version of the SLMS algorithm, which proved to yield significantly better training results for regression problems than the online version. The batch-mode LMS algorithm is given as

$$\Delta w_i = -\alpha \frac{\partial J}{\partial w_i} = \sum_{p=1}^{m} u_i^{(p)} \left(\hat{y}^{(p)} - y^{(p)} \right), \tag{5.16}$$

where J is the mean square cost function. Notice that in addition to performing analog multiplications, this algorithm requires the accumulation of analog values, which will be costly in hardware.

In contrast, the circuit design for the batch-mode SLMS algorithm (shown in Figure 5.7) uses only digital logic gates and comparators. The circuit consists of a global trainer and several individual synaptic trainers (one for each synapse). Input signals v_y and $v_{\mathbf{u}}$ are converted to the stochastic domain [58] by comparing them with independent uniformly-distributed random voltages v_r. For each training pattern, the SLMS algorithm evaluates the difference between the neuron output $v_{\hat{y}}$ and the expected system output v_y, and multiplies it by the system input v_{ui}. The result probabilistically increments, decrements, or holds the value of a counter, which will be used for synaptic weight adjustment. After all training patterns have been presented, a write voltage signal is applied to each of the synaptic CBRAM devices. The write voltage is either $+v_w$, $-v_w$, or 0, depending on the value stored in the counter and a threshold value θ_w.

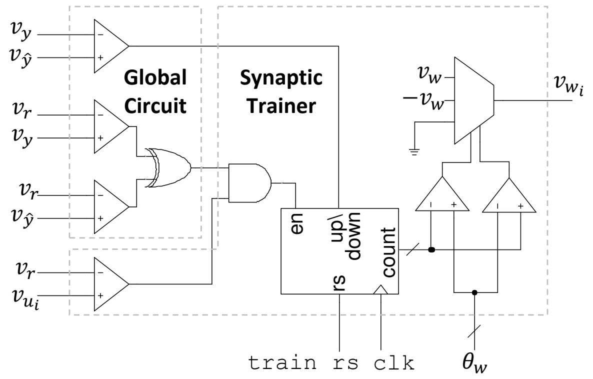


Figure 5.7: Batch SLMS circuit design.

Algorithm 1 summarizes the training scheme in more detail. First, training is enabled by setting `train_en`='1'. When this signal and the `train` signal are both high, the switch connected to v_w in the synapse circuit is closed (see Figure 4.18(a)), allowing a write voltage to be applied to the CBRAM devices. Lines 2-26 perform the batch-mode SLMS training algorithm described above. Initially, the `train` signal is set to '0' and `train_rs` is pulsed to reset all of the counters in each of the synaptic trainers. Then, lines 5-15 update the counters in the synaptic trainers. Note that the loop in lines 8-14 is actually unrolled in hardware, so each of the counters is updated in parallel. After the counters

Algorithm 1 Batch SLMS training algorithm.

```
1: train_en='1' to enter training mode
2: for $epoch$=1:$N_{epochs}$ do
3:     train='0'
4:     Pulse train_rs for one clock cycle to set $\texttt{count}_i$=0$\forall i$
5:     for $p$=1:$m$ do
6:         $\mathbf{u} = \mathbf{u}^{(p)}$
7:         $y = y^{(p)}$
8:         for $i$=0:$N$ do
9:             $v_{\hat{y}} = v_{\mathbf{u}} \cdot \mathbf{w}^{\mathrm{T}}/(N+1)$
10:            $U_i \sim \mathrm{B}\left(1, u_i^{(p)}\right)$
11:            $Y_i \sim \mathrm{B}\left(1, y^{(p)}\right)$
12:            $\hat{Y}_i \sim \mathrm{B}\left(1, \hat{y}^{(p)}\right)$
13:            $\texttt{count}_i += \mathrm{sgn}\left(y^{(p)} - \hat{y}^{(p)}\right) U_i \wedge \left(Y_i \oplus \hat{Y}_i\right)$
14:        end for
15:    end for
16:    train='1'
17:    for $i$=0:$N$ do
18:        if $\texttt{count}_i > \theta_w$ then
19:            Apply $+v_w$ V to synapse $i$
20:        else if $\texttt{count}_i < \theta_w$ then
21:            Apply $-v_w$ V to synapse $i$
22:        else
23:            Apply 0 V to synapse $i$
24:        end if
25:    end for
26: end for
27: train_en='0' to enter test mode
```

are updated, the train signal is set to '1' (line 16) and the write voltages are applied to the synapses (lines 17-26, which are also unrolled). Finally, training is disabled by setting train_en='0'.

The reason that the batch-mode SLMS algorithm works can be described intuitively as follows. If the error gradient for a particular weight is consistently large and positive over all of training patterns, then that weight is likely to be decreased. If the error gradient is consistently large and negative over all of the training patterns, then that weight is likely to be increased. Otherwise, for small error gradients, the synaptic weights are less likely to be modified. These changes allow the weights to descend the error gradient and settle at a global minimum. Mathematically, we can express the expected value of each synaptic write voltage (which relates to the expected change in synaptic weight) as

$$\mathrm{E}[v_{wi}] \approx \begin{cases} +v_w, & \mathrm{E}_i > \theta_w \\ -v_w, & \mathrm{E}_i < -\theta_w \\ 0, & \text{otherwise} \end{cases}, \tag{5.17}$$

where

$$\mathrm{E}_i \equiv \mathrm{E}\left[\sum_p v_{ui}^{(p)} \left(v_y^{(p)} - v_{\hat{y}}^{(p)}\right)\right] \tag{5.18}$$

and $\mathrm{E}[\cdot]$ is the expectation. An important free parameter that must be tuned in our system is θ_w. If θ_w is too small, then the algorithm will become very sensitive to noise in the training data and will not converge. Conversely, if θ_w is too large, the algorithm will converge before it reaches the global minimum. In general, the threshold value should be larger for larger training batch sizes because the expected count values in the synaptic trainers will be higher. In this work, we have used θ_w=5, which was determined empirically.

5.3 Min Algorithm

Another approach to training an NMS, shown in Figure 5.3, is to approximate complex functions such as multiplication with functions that are easier to implement in hardware. The proposed circuit first converts an input current to a pulse width (left schematic). Then, using two current-to-pulse width (itop) converters and an AND gate, one can compute the min. In this case, the pulse width of `we` will be proportional to $\min(i_{x_j}, |i_{x_i} - i_{\hat{x}_i}|)$. The resulting write voltage v_w. If the normalized values of $i_{\hat{x}_j}$ and $|i_{x_i} - i_{\hat{x}_i}|$ are both in the unit interval, then the normalized write voltage v_w will be very similar to their product.

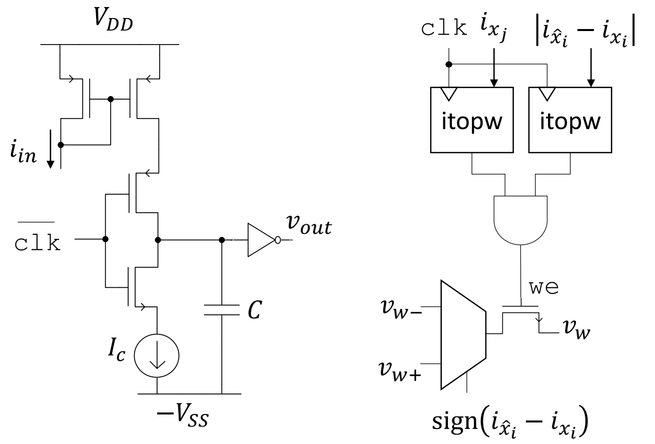


Figure 5.8: Circuits for implementing the min function.

The simplification presented above is used to design a learning circuit similar to the LMS rule [110], which can be written as

$$w_{ij} := w_{ij} + \alpha x_j x_{iD} = w_{ij} + \alpha x_j(\hat{x}_i - x_i). \tag{5.19}$$

Here, x_{iD} is the difference between the neuron's actual and expected outputs. In this work, a novel circuit for implementing a learning rule similar to (5.19). The modified training rule becomes

$$w_{ij} := w_{ij} + \alpha \text{sign}(x_{iD}) \min(x_j, |x_{iD}|). \tag{5.20}$$

To implement this in hardware, we first find $|x_{iD}|$ using a modified current subtraction circuit. We also find $\text{sign}(x_{iD})$ using a current comparator. Then, $i_{x_{iD}}$ and i_{x_j} are converted into pulse widths using the circuit in Figure 5.3. If the buffer's threshold voltage is low, then the length of the pulse width at v_{out} measured from the rising edge of `clk` to the falling edge of v_{out} will be

$$t_w \approx \frac{i_{in}}{I_c}\frac{T_{clk}}{2}, \tag{5.21}$$

where T_{clk} is the clock period. Combining two such circuits and an AND gate gives us the $\min$ function, which is used as a write enable, WE, signal for each synapse. Finally, the sign of the $i_{x_{iD}} = i_{\hat{x}_i} - i_{x_i}$ is used to select a positive or negative write voltage. The variation in current matching will have the largest effect on the functionality of the proposed training circuit.

5.4 Summary

This chapter has discussed primary design overheads in existing training algorithms for NMSs. New algorithms were designed to reduce the area and complexity of existing algorithms. Specifically:

- A novel stochastic training algorithm was developed to that reduces the area cost of the LMS algorithm by $\approx$3.5$\times$. The algorithm can operate in both online and batch modes.

- A novel circuit was designed to approximate multiplications in the unit interval as a $\min$ function, reducing the complexity of gradient calculations for NMS training.

Chapter 6

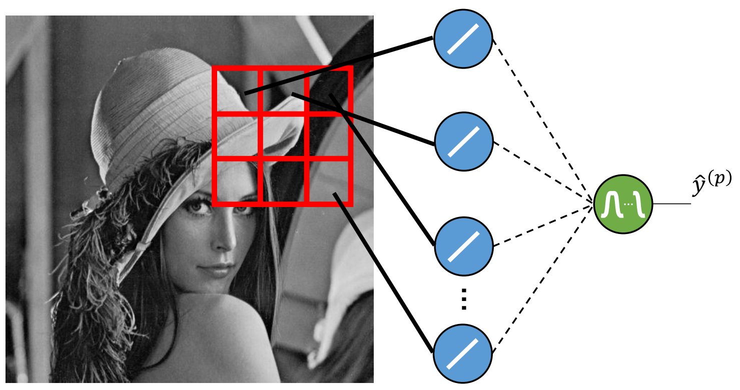
NMSs for Visual Information Processing

This chapter focuses on NMS designs for visual information processing tasks using the primitive synapse, neuron, and plasticity circuits discussed in the previous chapters. Here, the primary design challenge is to choose a network topology that meets constraints from both the algorithm and the circuit levels. In particular, the topology must facilitate a specific visual information processing task, and it should also be feasible to implement using primitive circuits. The vision tasks studied in this work include

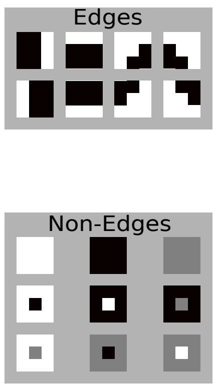
- Feature detection
- Classification
- Clustering

Each task corresponds to a different level in the brain's vision processing hierarchy. In addition, the topologies explored in this chapter are all previously unstudied within NMS design. Each was chosen reduce the cost of on-chip training. The rest of this chapter discusses the design of NMSs for the 3 information processing tasks listed above in detail.

6.1 Feature Detection



(a)



(b)

Figure 6.1: (a) Simulation setup for testing a single-layer, one-output neural network for an edge detection application. (b) Training patterns for edge detection (adapted from [111]).

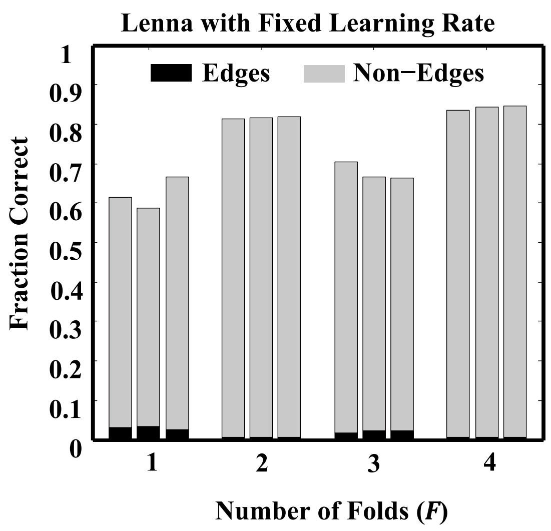
Detection of low-level features such as edges, lines, and corners is a critical step that takes place in the brain's early visual system [112]. Edge detection, in particular, is an important vision task that has wide utility in areas such as automated medical diagnosis [111] and is a difficult problem for neural networks. The difficulty stems from the fact that edge detection is a non-linearly separable problem. In fact, it is very similar to a parity or XOR problem, where an NMS (or any classifier) must determine if the number of '1' inputs is even or odd. It has been proven by Minsky and Papert [113] that this type of problem cannot be solved by single-layer neural networks with monotonic activation functions. However, if the activation function is not monotonic, as is the case for the folding amplifier design, then a single-layer network should be able to learn multiple decision boundaries.

This hypothesis was tested by designing a single-layer NMS to detect edges in grayscale images, as shown in Figure 6.1(a). Each of the NMSs 9 inputs connects to one pixel in a

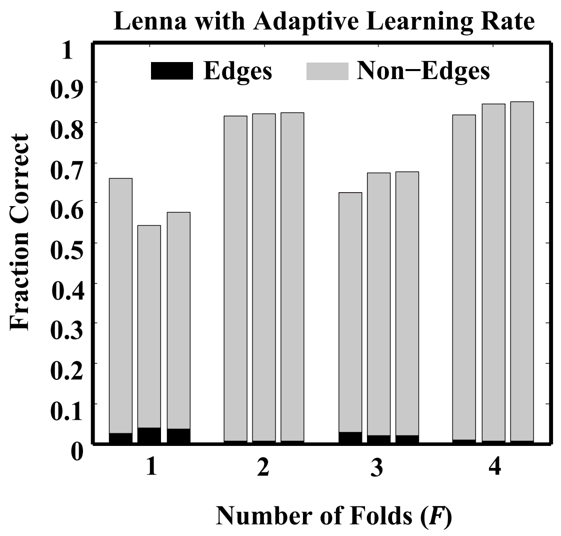
3-pixel by 3-pixel window, which scans across the entire input image. At each location, the network determines if the pixel in the center of the window is part of an edge. A threshold function is applied to the network's output to get a binary decision. The 17 patterns shown in Figure 6.1(b) [111] were used for supervised training. One training cycle consists of randomly initializing the weights, selecting a training pattern at random, applying it to the network's inputs, setting the expected network output to '1' in the case of an edge pattern or '0' in the case of a non-edge pattern, and applying the Perceptron learning rule.

The accuracy results are shown in Figure 7.2. Figures 7.2(a) and 7.2(b) show results for the Lenna image with adaptive and fixed learning rates, respectively. Similarly, Figures 7.2(c) and 7.2(d) show results for the Clock image with adaptive and fixed learning rates, respectively. These input images can be seen in Figures 6.3(a) and 6.3(f). In each plot, the fraction of correctly-classified pixels is plotted against the number of folds used for the output neuron. We determined whether or not our networks correctly classified pixels by comparing our output (edge-detected) images with an edge-detected image produced by MATLAB. For each fold factor, we plotted the results after 1, 2, and 5 training cycles, shown as the left, center, and right bar in each group, respectively.

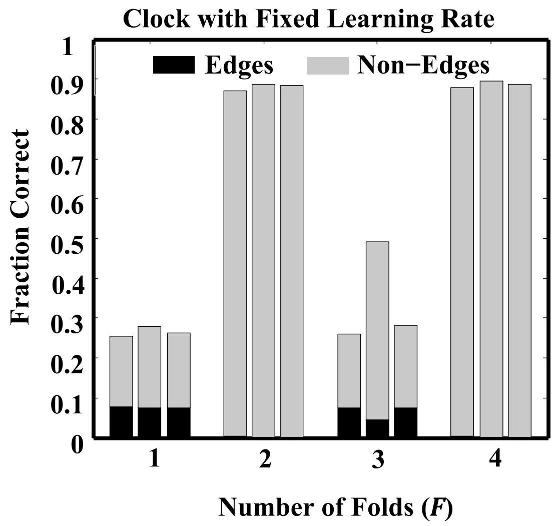
We hypothesized that we would see steady improvement in the fraction of correctly-classified pixels as the number of output neuron folds increases. Again, this is because each fold represents another decision boundary for the network. e.g., when $F = 4$, the network can theoretically learn functions with four decision boundaries. Our hypothesis is correct in the cases of one and four folds, which consistently perform the worst and best, respectively. For the Lenna image, the 4-fold neuron performs $\approx$20% better than the 1-fold neuron. The difference is even larger for the Clock image, where the 4-fold neuron



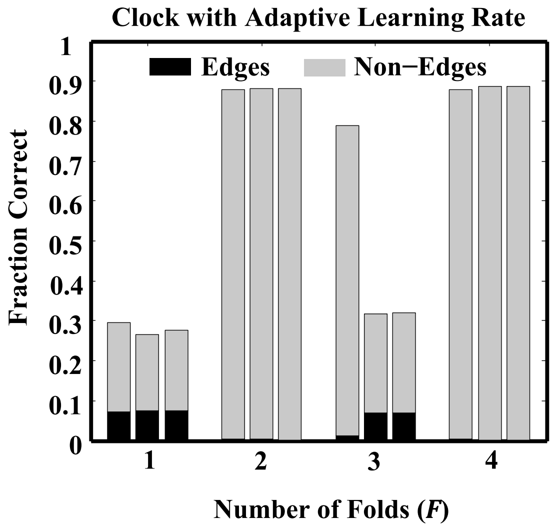
(a)



(b)



(c)



(d)

Figure 6.2: Quantitative comparison of edge detection simulation results: Lenna image with (a) adaptive and (b) fixed learning rates. Clock image with (c) adaptive and (d) fixed learning rates. Each plot shows the fraction of pixels correctly classified by the network for networks with 1, 2, 3, and 4-fold output neurons. Each group of three bars shows results, from left to right, after 1, 2, and 5 training cycles, respectively.

performs $\approx$65% better than the one 1-fold neuron. What we did not expect, however, is that the 2-fold neurons would perform almost as well as the 4-fold neurons and 3-fold neurons would perform almost as poorly as one fold neurons, consistently across all experiments. We are not sure if this is true in general (i.e. it may be application specific). However, it

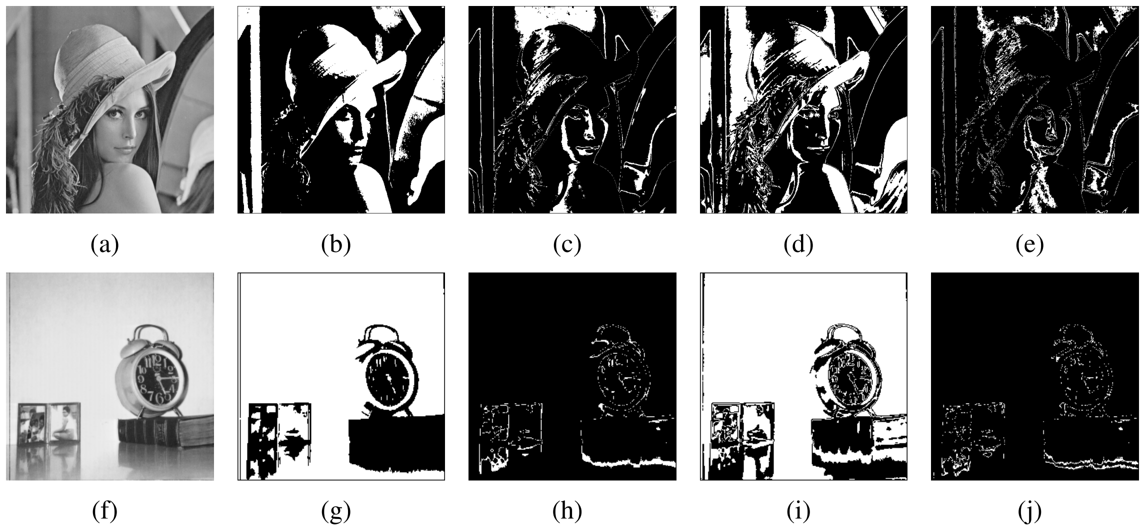


Figure 6.3: Qualitative comparison of edge detection simulation results: (a) Original Lenna image, and edge-detected images after 5 training cycles with a fixed learning rate and (b) 1-fold, (c) 2-fold, (d) 3-fold, and (e) 4-fold output neuron. (f) Original Clock image, and edge-detected images after 5 training cycles with a fixed learning rate and (g) 1-fold, (h) 2-fold, (i) 3-fold, (j) 4-fold output neuron [114].

tells us that care should be taken in choosing the type of activation function that can learn enough decision boundaries for a target application while keeping in mind that, in our case, neurons with more folds will have a higher area and power cost. Therefore, in this case, it would make more sense to use a 2-fold neuron, even though it doesn't perform quite as well as the 4-fold neuron.

A qualitative comparison of our results is shown in Figure 6.3. Figures 6.3(a) and 6.3(f) show the original grayscale images used for testing. Then, Figures 6.3(b) and 6.3(g), 6.3(c) and 6.3(h), 6.3(d) and 6.3(i), and 6.3(e) and 6.3(j) show the results after five training cycles with a fixed learning rate for the 1, 2, 3, and 4-fold neurons, respectively. One can immediately see the effect of only having one decision boundary in the case of the 1-fold neurons. Instead of classifying pixels as edges or non-edges, the network classifies them as belonging to light or dark regions of the image, which is a linearly-separable function. In the cases of the 2 and 4-fold neurons, the network is able to classify many of the different types of edges and non-edges that exist within the original images.

6.2 Classification

Object classification is a critical task in visual information processing. In the primate brain, objects are classified in the inferotemporal cortex, which receives information from the feature extraction layers in the primary visual cortex (e.g. the V1 cortex). A key parameter when designing an NMS for object classification is the dimensionality of the input space. When the number of dimensions is low, accuracy is improved by first increasing the dimensionality and then training on the higher-dimensional space. In contrast, when the number of dimensions is high, accuracy (especially generalization) can often be improved through dimensionality reduction techniques. More importantly, reducing the number of dimensions also reduces the size of the NMS, which improves area and energy efficiency. Both of these ideas are explored (increasing and reducing the dimensionality of the input space) in this work.

6.2.1 MNIST Dataset

An NMS with an MLP topology was designed for classifying MNIST images, which are handwrittend images of the numbers 0-9. To reduce the cost of training, only the output layer is adjusted (using online SLMS), while the hidden layer is untrained, providing a random projection to a higher dimensional space. This approach has been explored previously in software [90, 115, 116]. However, this work is the first to apply the idea in an NMS. The

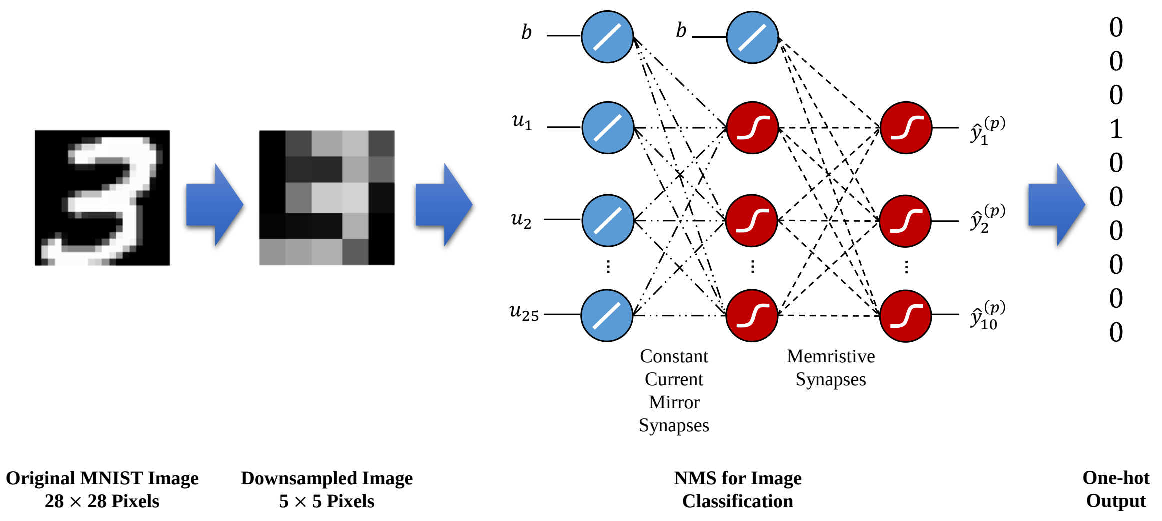


Figure 6.4: Simulation setup for classification on the MNIST database. The original MNIST images are reduced to 5×5 pixels and fed into the NMS. The NMS's output layer is trained using online SLMS, and a winner-take-all output is used to generate the hypothesis.

fixed, random weights in the hidden layer are implemented using constant current mirror synapses. The outputs of the hidden layer are the inputs to 10 ($M = 10$) logistic classifiers. Each of the weights for the logistic classifiers are implemented using a bipolar weight memristive synapse. In addition.

The simulation setup is shown in Figure 6.4. 1000 examples were used for both the training and test sets. Each sample is first reduced to 5×5 pixels. Then, the average values are scaled between 0 and 1. These values are used as the inputs to the network. The inputs are randomly projected to a 50-neuron hidden layer, and the output layer is trained using online SLMS. The final output is taken using winner-take-all to create a one-hot encoding of the hypothesized class.

The results are shown in Figure 6.5. Figure 6.5(a) shows the classification accuracy of the logistic classifiers versus training epoch for a single run, where α=0.001. Both algorithms have similar convergence in this case. The MSE versus learning rate is plotted

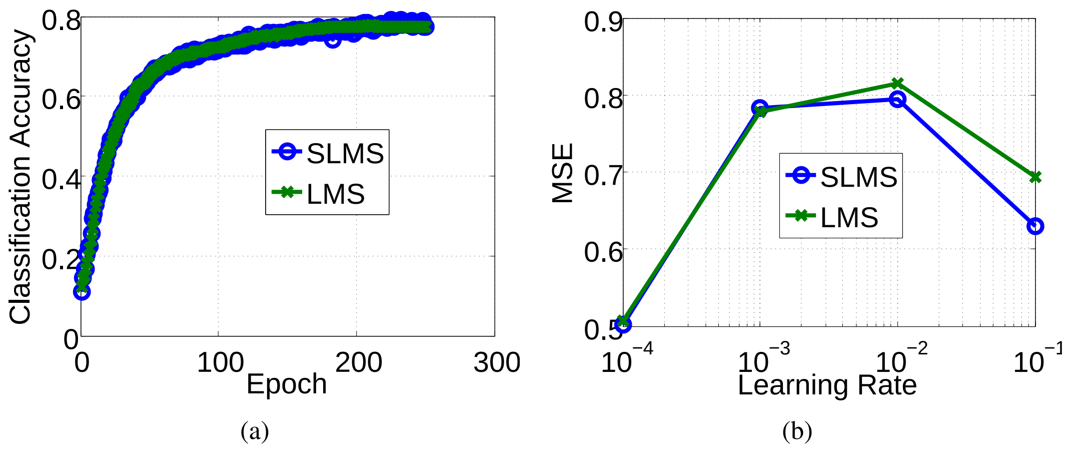


Figure 6.5: (a) Classification accuracy vs. epoch and (b) classification accuracy vs. learning rate for the logistic regression problem.

in Figure 6.5(b). For small learning rates, the accuracy is diminished. This is likely because the number of training epochs is limited to only 250. Also, for large learning rates, the accuracy starts to fall off because the algorithms are unable to fine tune. Notice that the SLMS algorithm is able to achieve accuracies similar to the LMS algorithm. Using the expressions for the area costs of SLMS and LMS, one finds that the LMS algorithm has $\approx 3.5\times$ larger area overhead than SLMS.

6.2.2 Caltech101 Dataset

This work also studied the effect of dimensionality reduction on an NMS's classification accuracy and area cost. Dimensionality reduction occurs in the early stages of visual processing in the primate brain. The goal is to map vectors $\mathbf{u}^{(p)} \in \mathbb{R}^O$ in a high-dimensional space to vectors $\pi^{(p)} \in \mathbb{R}^o = \mathbf{Lu}$, where $o \ll O$. Here, L is an $o \times O$ matrix, which can be found using a variety of techniques. In this work, $\mathbf{L}$ is found using locality preserving projections (LPP) [117]. The LPP dimensionality reduction step captures 99.9% of the variance in the original space. This particular study used LPP in conjunction with a

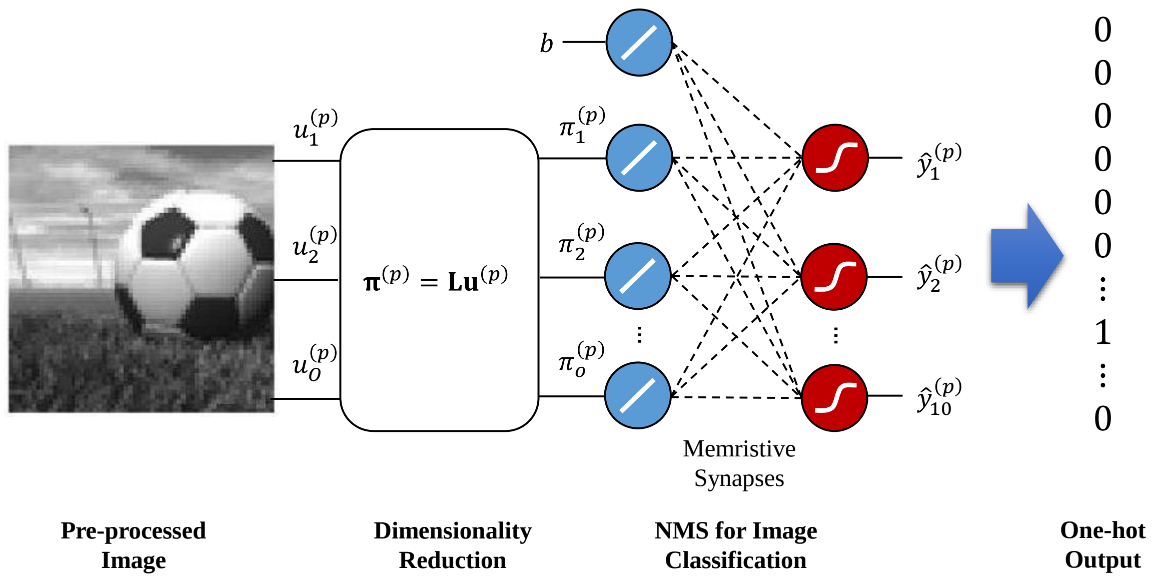


Figure 6.6: Simulation setup for classification on the Caltech101 database. The original (color) images are first converted to black and white and normalized. Then, their dimensionality is reduced using LPP. The lower-dimensional vectors are classified using a single-layer perceptron NMS.

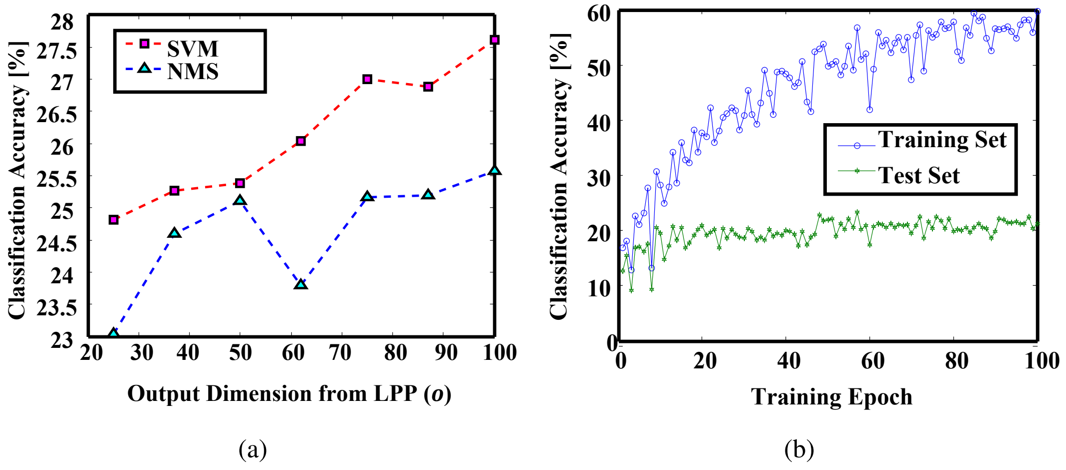


Figure 6.7: (a) Classification accuracy of linear SVM and single-layer neuromemristive architecture as a function of the top d dimensions from LPP. (b) Learning curves for the NMS using SLMS.

single-layer NMS to classify images from the Caltech101 dataset [118].

A diagram of the simulation setup is shown in Figure 6.6. The Caltech101 dataset contains between 40 and 800 sample pictures of objects from 101 categories. Each image was converted to black and white and normalized. Then, a dimensionality reduction matrix

L was found using LPP. The lower-dimensional samples were classified using a single-layer NMS with SLMS (α=0.0001). 5-fold cross-validation was used, whereby each of the 5 training/test splits used 80% of the dataset for training and 20% for testing. Figure 6.7(a) shows the mean classification accuracy versus o (the number of dimensions from the dimensionality reduction) for both the NMS network and a suppport vector machine (SVM). The training accuracies were ≈70% for the NMS and ≈90% for the SVM. On average, the NMS performs only 1.5% worse than the SVM.

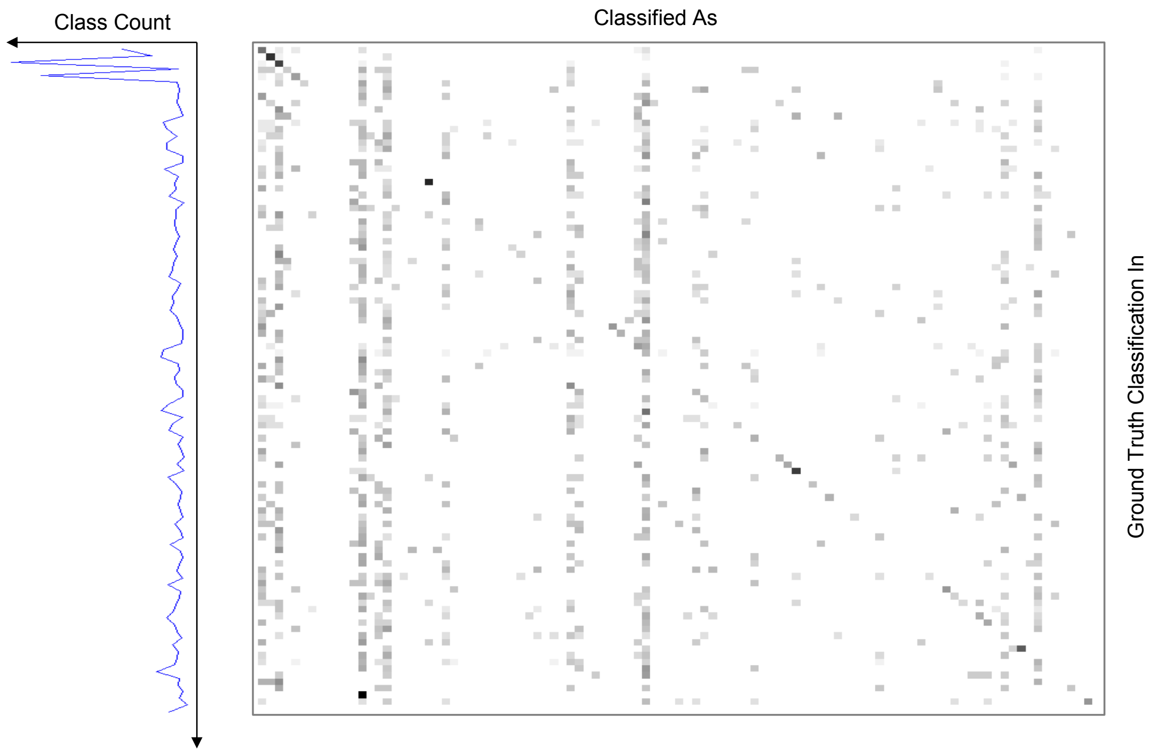


Figure 6.8: (left) Number of training samples per class; (right) Confusion matrix showing input ground truth class on the rows, and corresponding mapping on the columns.

Figure 6.8 shows the confusion matrix, along with the relative number of training samples used for each class in the dataset. For high classification accuracy, we should see a prominent diagonal and few gray squares off of the diagonal. Although the diagonal is

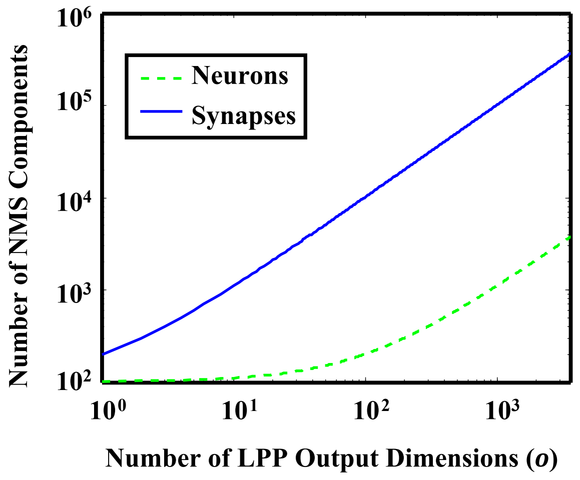


Figure 6.9: The NMS network size vs. the number of LPP output dimensions o.

clearly present, there are many off-diagonal non-zero (not white) values, indicating that the NMS found many common features among different object classes. Also note that the classes corresponding to the highest classification accuracy (darkest) also have the largest number of samples in the training set.

In this work, the dimensionality reduction step allows the size of the NMS classifier to be reduced significantly. This idea is illustrated in Figure 6.9, where we have plotted the number of NMS components (i.e. neurons and synapses) versus the number outputs of the LPP dimensionality reduction o. Before the dimensionality reduction step, the input images have $O = 60 \times 60$ pixel values. Feeding these directly into the NMS would require $\approx 3.7 \times 10^3$ neurons and $\approx 3.6 \times 10^5$ synapses. However, when o is small, e.g. 100, there is $\approx$97% reduction in the size of the NMS needed for classification.

Figure 6.10: The Caltech101 dataset. Two example images from each of the 100 classes is shown, after resizing, normalization, and conversion to grayscale.

6.3 Clustering

6.3.1 Overview

Clustering algorithms uncover structure in a set of m unlabeled input vectors $\{\mathbf{u}^{(p)}\}$ by identifying M groups, or clusters of vectors that are similar in some way. In one common approach, each cluster is represented by its centroid, so the clustering algorithm is reduced to finding each of the M centroids. This can be achieved through a simple competitive learning algorithm: Initialize M vectors $\mathbf{w}_i$ by assigning them to randomly-chosen input vectors. These will be referred to as weight vectors. Then, for each input vector, move the closest weight vector a little closer. After several iterations, the algorithm should converge with the weight vectors lying at (or close to) the centroids. Of course, there are several parameters which must be defined, including a distance metric for measuring closeness. The most obvious choice is the ℓ^2-norm. However, computing this is expensive in terms of hardware because it requires units for calculating squares and square roots. In addition, as we will discuss later, it is easy to use a high-density memristor circuit called a crossbar to compute dot products between input and weight vectors. Therefore, it is preferred to use a dot product as a distance metric. For example, if all of the vectors are normalized ($\|\mathbf{u}^{(p)}\| = \|\mathbf{w}_i\| = 1$), then $\mathbf{w}_{i*} \cdot \mathbf{u}^{(p)} > \mathbf{w}_i \cdot \mathbf{u}^{(p)} \forall \mathbf{w}_i \neq \mathbf{w}_{i*}$, where $\mathbf{w}_{i*}$ is the closest weight vector to $\mathbf{u}^{(p)}$. However, the constraint that $\|\mathbf{u}^{(p)}\| = \|\mathbf{w}_i\| = 1$ creates a large overhead, because every input vector has to be normalized and every weight vector has to be re-normalized each time it is updated.

We propose the following solution: Map each input vector to the vertex of a hypercube centered about the origin: $\mathbf{u}^{(p)} \in \{-1, 1\}^N$, where N is the dimensionality of the input space. Now, $\mathbf{w}_i \cdot \mathbf{u}^{(p)}$ will yield a scalar value $d^*_{i,p}$ between $-N$ and $+N$. Moreover, this scalar value can be linearly transformed to a distance $d_{i,p}$ which is the ℓ^1-norm, or

Manhattan distance, between the weight vector and the input:

$$d_{i,p} \equiv N - d^*_{i,p} = \sum_{j=1}^{N} |w_{i,j} - u_j^{(p)}|. \quad (6.1)$$

Using this distance metric, we don't ever need to re-normalize the weight vectors. Furthermore, mapping input vectors to hypercube vertices can usually be accomplished by thresholding. For example, grayscale images can be mapped by assigning -1 to pixel values from 0 to 127 and +1 to pixel values from 128 to 255. Algorithm 2 summarizes the algorithm. The first two lines are initialization steps. Within the double `for` loop x_i is 1 when i corresponds to the index of the closest vector (called the winner) and 0 otherwise. Then, the weight components of the winner are moved closer to the current input vector using a Hebbian update rule. The pre-factor α, which is called the learning rate, determines how far the weight vectors move each time they win. Notice that this algorithm is completely unsupervised, so there are no labeled input vectors.

Algorithm 2 Proposed clustering algorithm.

```
1: Map inputs to hypercube vertices.
2: Initialize weight vectors to random input vectors.
3: for $epoch$ = 1:$N_{epochs}$ do
4:     for $p$ = 1:$m$ do
5:         $d^*_{i,p} = \mathbf{w}_i \cdot \mathbf{u}^{(p)} \;\; \forall i = 1, 2, \ldots, M$
6:         $x_i = \begin{cases} 1, & d^*_{i,p} = \max(d^*_{i,p}) \\ 0, & \text{otherwise} \end{cases} \forall i = 1, 2, \ldots, M$
7:         $\Delta w_{i,j} = \alpha x_i u_j^{(p)} \;\; \forall i = 1, 2, \ldots, M \;\; \forall j = 1, 2, \ldots, m$
8:     end for
9: end for
```

The unsupervised clustering algorithm discussed above can be implemented efficiently in an NMS by representing weight vectors as memristor conductances. A block diagram of the proposed design is shown in Figure 6.11. The inputs, which are represented as positive and negative currents, are fed through M crossbar circuits using the crossbar-based memristive synapse designed in this work. Each crossbar computes the distance between the current input and the weight vector represented by its memristors' conductances.

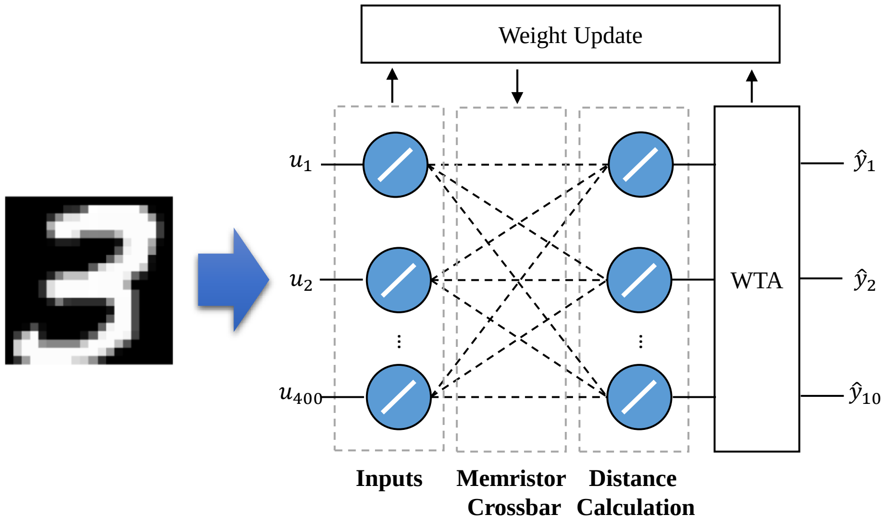


Figure 6.11: Block diagram of proposed NMS for unsupervised clustering.

So far, only distance calculation part of the algorithm has been discussed 6.11 (line 5 in Algorithm 2). The winner-takes-all circuit (line 6 in Algorithm 2) can be implemented in a number of ways. In this work, we used the current-mode design described in [119]. Finally, the weight update (line 7 in Algorithm 2) can be computed using simple combinational logic circuits.

6.3.2 Clustering MNIST Images

One exciting application of the proposed hardware is automatically identifying clusters in sets of images. We took 1000 images (m=1000) from the MNIST handwritten digit dataset and clustered them using a behavioral model of the NMS described in the last section. Each

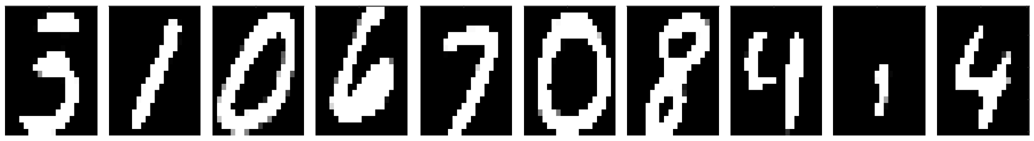


Figure 6.12: 10 cluster centroids found in a set of 1000 MNIST images using the proposed NMS.

image was originally 20×20 grayscale pixels (N=400). They were mapped to hypercube vertices using the thresholding approach discussed earlier. In addition, we used 10 clusters (M=10), 500 training epochs (N_{train}=500), and α=0.005. The results are shown in Figure 6.12. Here, we have plotted the weight vectors representing the centroid of each cluster. Figure 6.13 shows the cost versus the training epoch, where the cost is defined as

$$J = \sum_{p=1}^{m} \left(\min\, d_{i,p} \forall i\right). \tag{6.2}$$

The cost function for the proposed NMS approaches that of MATLAB's built-in k-means clustering after 500 epochs.

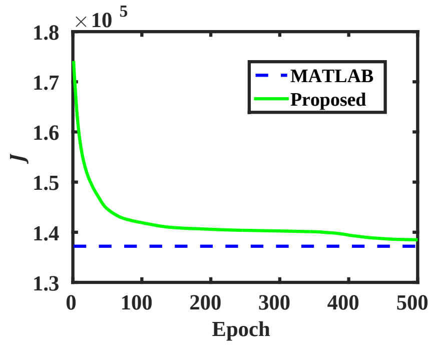


Figure 6.13: Cost function versus epoch while clustering MNIST images using the proposed NMS.

6.4 Summary

This chapter discussed 3 visual information processing tasks that can be implemented with NMSs built from the primitive circuits designed in this work. The novel contributions presented in this chapter are

- Demonstration of edge detection in a single-layer NMS using a novel periodic activation function circuit.
- Demonstration of the functionality of the memristive synapse and SLMS algorithm designed in this work for classification in both multi-layer and single-layer NMSs.
- Design of a novel memristor crossbar-based approach for image clustering that achieves accuracy similar to the k-means algorithm

Chapter 7

Effects of Device Variations on System-Level Performance

The NMSs designed in the last chapter were simulated assuming nominal device and circuit characteristics. However, it was shown in Chapters 3 and 4 that both MOSFET and memristor variations will lead to large variations in synapse and neuron circuit behavior, which are further exacerbated when operating in the subthreshold region. Therefore, it is critical to study the effects of device-level variations on system-level performance. As far as the author is aware, this is the first study to include active (i.e. MOSFETs) and passive (i.e. memristors, resisitors, and capacitors) device variations in a system-level NMS simulation. In the first part of this chapter, the voltage-mode regression circuit discussed in Chapter 4 is applied to an electrical load forecasting problem. On-chip batch-mode SLMS is used to train CBRAM-based synapses. As a result of on-chip training, high accuracy can be achieved with large device-level variations. The later part of this chapter discusses the effects of device variations when off-chip training is used. A partial on-chip training method is developed to reduce achieve a balance between the pros and cons of off-chip training.

7.1 Case Study: NMS for Electrical Load Forecasting

An NMS based on the CBRAM synapse and voltage-mode neuron circuits was designed to study the effects of device-level variations on system-level performance. The NMS performs linear regression on a set of datapoints $\{(\mathbf{u}^{(p)}, y^{(p)})\}$, where $\mathbf{u}^{(p)} \in \mathbb{R}^{N+1}$ (N-dimensional inputs with an additional bias term) and $y^{(p)} \in \mathbb{R}$. The overall system block diagram is shown in Figure 7.1. During training (training mode), each input vector **u** and output scalar y are pre-processed to convert them from their original representations (e.g. binary) to voltages $v_{\mathbf{u}}$ and v_y that range from 0 to V_{DD}. Then, the synapses $S_0, S_1, \ldots, S_N$ and neuron circuit are used to evaluate the scaled dot product $v_{\hat{y}} = v_{\mathbf{u}} \cdot \mathbf{w}^{\mathrm{T}}/(N+1)$, where $\mathbf{w} = [w_0, w_1, \ldots, w_N]$ is a weight vector. Each w_i corresponds to one of the synapse circuits S_i. The circuit operates in either training or test mode. In training mode, the batch SLMS training algorithm uses the error between v_y and actual outputs $v_{\hat{y}}$ to update weight values. After the training process is complete, the SLMS training circuit is disabled and the NMS enters test mode. In test mode, the synaptic weights remain constant, and the system predicts the output $v_{\hat{y}}$ corresponding to new input vectors using a linear fit. The post-processing step transforms the voltage representation of the output to another form such as binary. It may also shift and scale the output depending on the application.

The NMS described above was applied to electrical load forecasting in smart grids using an autoregression model. Accurate load prediction in smart grids is critical in order to efficiently manage them and minimize their failures. Before The NMS has 3 inputs (N=2 plus 1 bias input). The training and test data are composed of hourly power consumption values (in MW) from a power grid (located in US Mid-Atlantic Region) [120], collected for the month of January, 2012 and January, 2013, respectively. The goal of the system was to forecast the power consumption for the hour $t+1$ given the power consumption values from hours t, $t-1$, and $t-2$. More specifically, the inputs to the system during training were of the form $\mathbf{u} = [b, P_t - P_{t-1}, P_{t-1} - P_{t-2}]$, where b is a the constant bias term, and P is the power consumption. Each of the 3 synapses used 20 CBRAM devices (10 excitatory and 10 inhibitory). During training, the expected output $y = P_{t+1}$ is also supplied as an input. The total learning duration for the experiment was ≈50 μs (500 epochs).

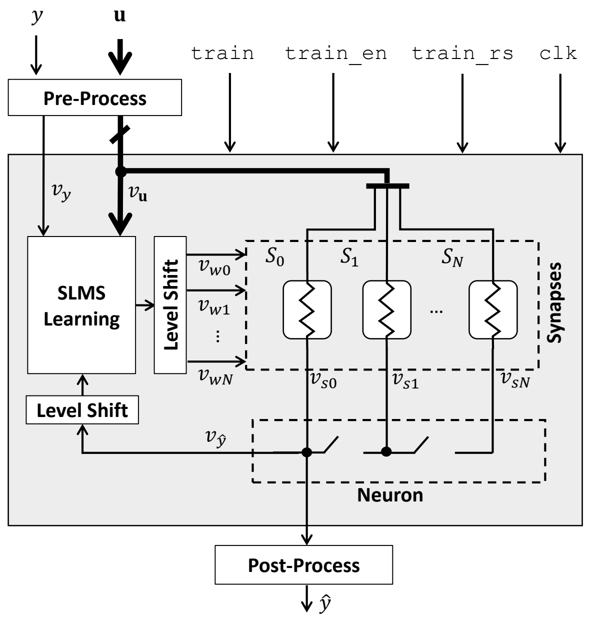


Figure 7.1: Block-level diagram of the proposed NMS for linear regression using the SLMS training algorithm.

7.1.1 Simulation with Nominal Parameters

The nominal values of the simulation parameters for the NMS are shown in in Table 7.1. These values were used to determine the baseline system accuracy. Figure 7.2(a) shows the actual and predicted electrical loads vs. time for the test data in 8-hour increments. Before training, we observe a large error in both the offset and slope of the predicted load.

Table 7.1: Nominal system parameters used for electrical load forecasting.

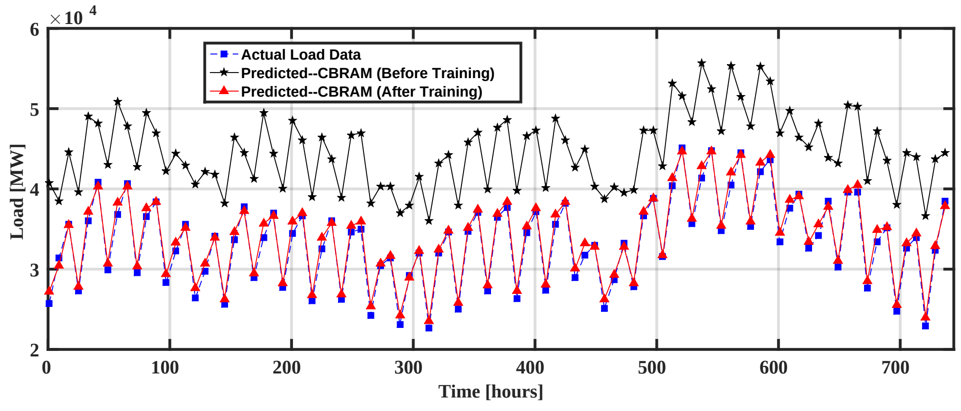
Parameter/Metric	**Value**
Simulation Parameters	
t (Capacitance charge time)	100 ns
CBRAM devices per synapse	20
N+1	3
Training epochs	500
$\|v_w\|$	1.5 V
t_w (Time that write voltage is applied)	500 ns
p_{switch}	0.05
Training time	50 μs
Simulation Metrics	
Mean accuracy	96%
Peak accuracy	97.5%
Energy per training epoch	5.4 mJ
Power	15 μW
Area	14.5 μm^2
CBRAM switch events	150

After training, the predicted and actual loads are very similar. Figure 7.2(b) gives a more detailed look at the data from the first day in January 2013. In addition to the forecasted and actual load values, we've also plotted a forecasted value assuming an *ideal* synapse. The ideal synapse can attain any (continuous) synaptic weight value. The weights in this case were found by solving the normal equation for the associated linear system. Notice that the forecasted load after training is very close to the ideal fit.

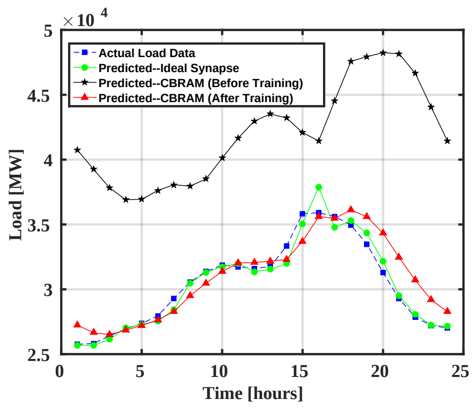
Figure 7.2(c) shows the accuracy for the test data versus training epoch. The accuracy is calculated as:

$$Accuracy = 100\% \left(1 - \frac{1}{m}\sum_{p=1}^{m}\left|\frac{y^{(p)} - \hat{y}^{(p)}}{y^{(p)}}\right|\right), \tag{7.1}$$

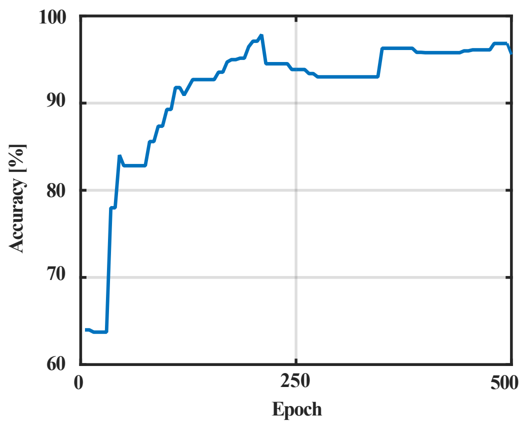
where m is the number of test patterns (number of predictions made on the January 2013 test data). Notice that the trend is generally increasing, but occasionally decreases due to the stochastic nature of the SLMS training algorithm. The curve in Figure 7.2(c) is for one



(a)



(b)



(c)

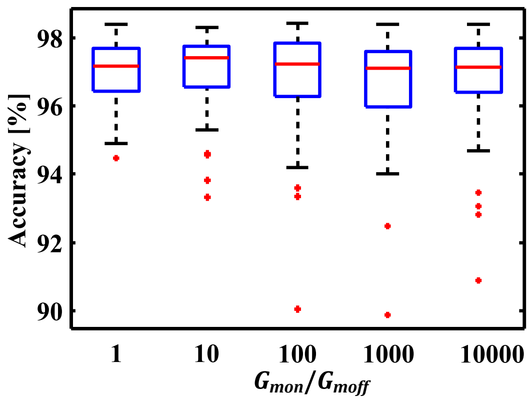
Figure 7.2: (a) Predicted load vs. time for the month of January 2013 in 8-hour increments. (b) Predicted load vs. time for January 1, 2013 (24 hours). (c) Load prediction accuracy versus training epoch.

simulation run. However, all of the simulations discussed in this section were run 10 times and averaged.

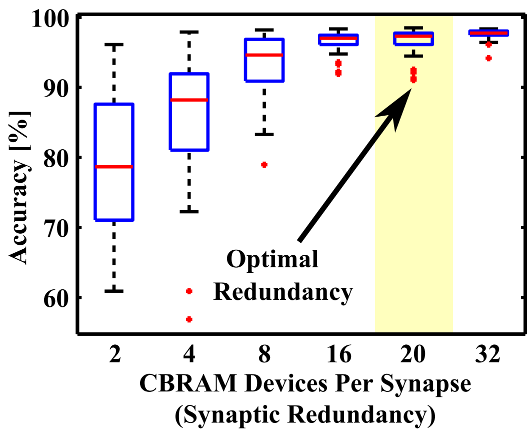
The simulation metrics for the nominal parameters are listed at bottom of Table 7.1. After training, the mean accuracy approaches 96%. In comparison, the ideal fit reaches an accuracy of 98%. The energy consumed per training epoch was 5.4 mJ, and the overall system power consumption was 15 μW. We estimated the area of our design to be 14.5 μm^2. An additional metric that is important in our system is the number of CBRAM switching events. Memristive devices, including CBRAM, have a limited endurance (number of times they can be switched before they become stuck in one state). Therefore, it is critical to limit the number of switching events in order to maximize the lifetime of the system. A deterministic online training algorithm would induce approximately 1 switching event per synapse per training pattern per epoch. For our system, this amounts to $\approx 1.86\times10^4$ switching events per device during training. In contrast, our batch-mode SLMS training algorithm reduces the number of switching events per device to ≈ 2.5. If we assume the system lifetime is directly proportional to CBRAM switching events, we see that the proposed training algorithm increases the system lifetime by over 7000$\times$.

7.1.2 Variation Analyses

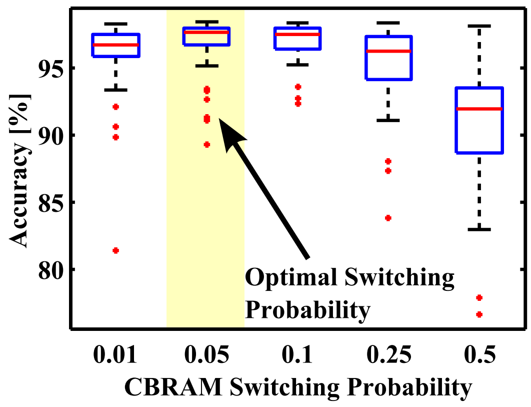
In addition to the nominal simulations discussed above, we have also investigated the effects of different device and parameter variations on the system's forecasting accuracy. The results are summarized in Figure 7.3. Figure 7.3(a) shows the impact of CBRAM programming window (G_{mon}/G_{moff}) on accuracy. The programming window can be modulated by controlling compliance current (maximum current allowed to pass through the devices) as demonstrated in [75]. The system shows good immunity to variations in the device conduction ratio. Surprisingly, the system accuracy is still good even when G_{mon}/G_{moff}=1. At first, this seems counterintuitive, because a conductance ratio equal to unity implies that each synapse will have one fixed weight state. However, the log-normally-distributed random variations in the CBRAM *on* and *off* state conductances allow the synaptic weights to reach multiple states even if the mean ratio is unity. In fact, as a result of the wide distribution of the state conductances, we can consider each CBRAM device as having quasi-continuous conductance states that can be reached probabilistically.



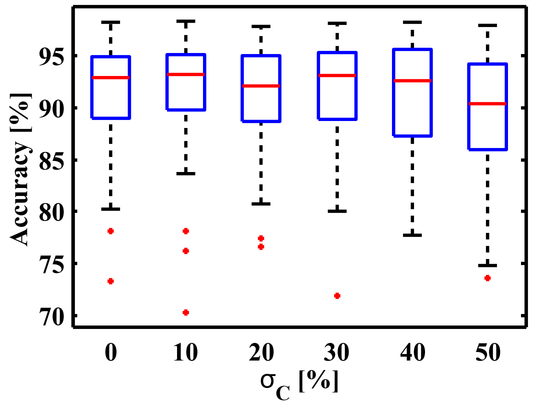
(a)



(b)



(c)



(d)

Figure 7.3: Impact of device and parameter variations on the system forecasting accuracy. (a) Impact of variations in the CBRAM programming window (G_{mon}/G_{moff}) (b) Forecast accuracy vs. the number of CBRAM devices used in each synapse. (c) Forecast accuracy vs. CBRAM switching probability. (d) Forecast accuracy vs. σ_C (variation in the capacitors used in the synaptic circuits).

An important design parameter is the number of CBRAM devices used per synapse k. Fewer devices leads to reduced area. However, using more devices allows each synapse to have higher weight resolution. The results are shown in Figure 7.3(b). We observe an asymptotic increase in the accuracy as the number of devices increases. In fact, one can show that as the number of CBRAM devices per synapse becomes large, then the forecast accuracy will approach the accuracy of the ideal fit (98%). It is also worth noting that the run-to-run variance of the accuracy becomes smaller with increased synaptic redundancy.

We have empirically chosen a synaptic redundancy of 20 as the optimal value as it exhibits high accuracy, low variance, and uses relatively few devices (e.g. compared to 32).

Another critical design parameter is the CBRAM switching probability. In Figure 7.3(c), we show the forecasting accuracy vs. p_{switch}. The shape of the curve created by the mean accuracies is concave, with the optimal (highest mean accuracy) occurring at a 0.05 switching probability. At lower switching probabilities, the system takes a long time to converge, so 500 training epochs are not enough to reach a maximum accuracy. On the other hand, as the switching probability becomes large, it is difficult for each synapse to achieve high resolution weight values because several CBRAM devices may switch concurrently. Therefore, a low switching probability (< 0.25) is desired for $> 90\%$ test accuracy. As discussed in Section 3.2.2, the switching probability can be controlled by adjusting the flux applied to the CBRAM devices during the write process.

Figure 7.3(d) shows the impact of synapse capacitance variation. The accuracy is fairly consistent when the capacitance variations are large (e.g. 50%), demonstrating the robustness of the proposed system. This is a critical property, as it is generally difficult to control capacitance values within 30% error. This result also demonstrates a general property of neuromoprhic systems, which we have explored in detail in [121]. That is, the system is able to incorporate and compensate for device-level variations during the learning process making it inherently robust.

7.2 Off-Chip Training

Training NMSs presents a formidable challenge due to several CMOS and memristor process variations. On-chip training is very effective at overcoming these, as demonstrated in the last section. However, it is not feasible to implement many sophisticated training algorithms on-chip. On the other hand, off-chip training has the advantage of flexible software training algorithms. A high-level overview of off-chip training for an NMS is illustrated in Figure 7.4. An ideal model of the network is trained off-chip using a software training algorithm (*e.g.* backpropagation, resilient backpropagation, Levenberg-Marquardt, genetic algorithms, etc.). Then, data from the trained model are used to train the on-chip NMS. Choosing which data (the weight matrix, the training vectors, the expected outputs of individual neurons, etc.) to incorporate into this training process is challenging, especially when the on-chip NMS is affected by process variations.

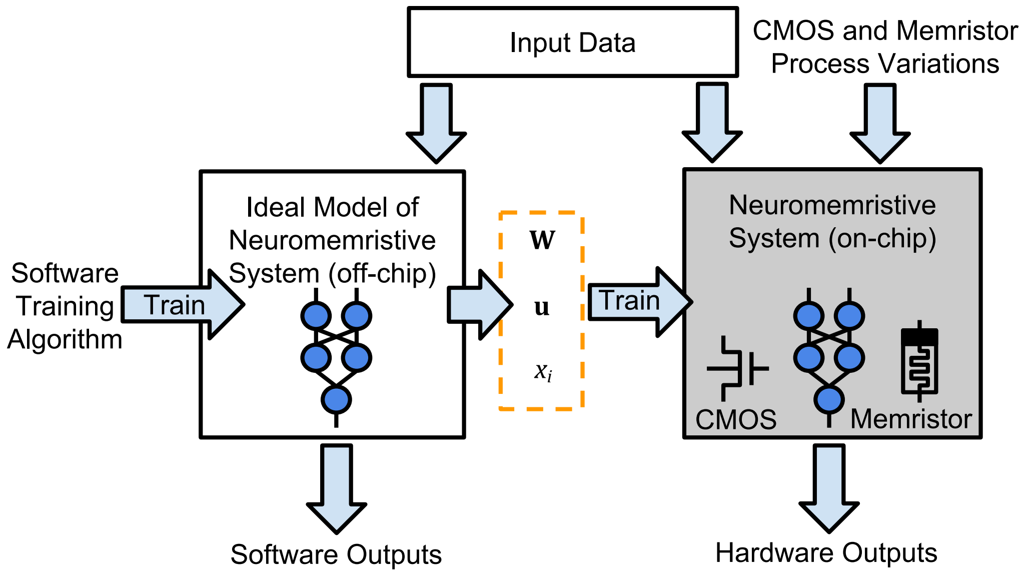

Input Data
CMOS and Memristor Process Variations
Software Training Algorithm
Train
Ideal Model of Neuromemristive System (off-chip)
W
u
x_i
Train
Neuromemristive System (on-chip)
CMOS
Memristor
Software Outputs
Hardware Outputs

Figure 7.4: High-level depiction of off-chip training for an NMS.

This research presents two fundamentally different approaches to training NMSs off-chip: *weight programming* and *feature training*. The first step in both methods is to train an ideal (no process variations) software model of the NMS. In the weight programming method, the weight matrix $\mathbf{W}$ is directly transferred onto the chip. That is, each weight value in the hardware NMS is programmed to match the corresponding value in the ideal NMS model. This method is the most straight-forward, and has been successfully demonstrated in [122] for an NMS perceptron (single-layer neural network). In the feature training method, each neuron in the NMS is trained (using our proposed training rule) sequentially, using their expected outputs from the trained software model. In both training approaches, analog and digital interface circuitry are needed including shift registers, logic gates, counters and current comparators, and sample and hold circuits. The interface circuitry overhead is similar for both weight programming and feature training.

7.2.1 Weight Programming

The proposed weight programming algorithm is shown in Algorithm 3. First the weight matrix $\mathbf{W}$ is found by training the ideal NMS model using e.g. resilient backpropagation algorithm. Then, each valid weight value (corresponding to a non-zero entry in the adjacency matrix $\mathbf{A}$) is transferred onto the chip. The address of the corresponding hardware synapses is found using an address mapping Addr and programming is enabled, allowing write voltages to be applied to the memristors in the synapse. Note that the output node of the synapse is grounded during programming. After a specified number of epochs N_{epochs}, the process repeats for the next weight value.

Algorithm 3 Weight programming

```
W =weights from off-chip software training
for all $i = 1 : N_x$ do
  for all $j = 1 : N_x$ do
    prog_en='0'
    if A$(i, j)$ then
      addr=Addr$(i, j)$
      $w_{addr} = w_{ij}$
      prog_en='1'
      Wait for $N_{epochs}T_{clk}$
    end if
  end for
end for
```

The total training time for weight programming is estimated as

$$t_{wp} \approx N_w T_{clk} N_{epochs} \tag{7.2}$$

where N_w is the total number of weights to be programmed. The area of the NMS, excluding the interface circuitry, can roughly be estimated as

$$\frac{A_{wp}}{WL} \approx ts[3(2+N) + 7(N_h + M) + 3N_w] \tag{7.3}$$

where ts is a transistor size factor, which is nominally 1. The constants N, N_h, and M are the numbers of inputs, hidden-layer (middle layer) neurons, and the number of outputs, respectively.

7.2.2 Feature Training

In the weight programming scheme, there was no actual training performed in hardware. In contrast, feature training treats each of the neurons in the network as a single-layer perceptron. Each one is trained using our proposed training rule, starting with those in the hidden layer. The algorithm is shown in Algorithm 4. First, the ideal model of the network is trained off-chip. Then, each on-chip sigmoid neuron i is trained for N_{epochs}. Here, $N_x = N_h + M$. Within each epoch, the algorithm loops through all of the training vectors $\mathbf{u}$. An address signal is set to isolate the synapses that are inputs to the i^{th} neuron. Then, a training enable signal is set and the neuron is trained for lr clock cycles. The lr constant, which is a positive integer, is related to the learning rate α as

$$\alpha = (V_{DD} + V_{SS}) lr T_{clk} I_{max} / (2 I_c). \tag{7.4}$$

Algorithm 4 Feature training

```
Train ideal network model off-chip
for all $i = 1 : N_x$ do
    for all $epoch = 1 : N_{epochs}$ do
        for all $\mathbf{u}$ do
            train_en='0'
            addr=Addr($i$)
            Set $\hat{x_i}$ from trained network
            Set $\mathbf{u}$ as the input
            train_en='1'
            Wait for $lrT_{clk}$
        end for
    end for
end for
```

The total training time for feature training is estimated similar to the weight programming training time as

$$t_{ft} \approx N_x N_{epochs} N_{\mathbf{u}} T_{clk} lr. \tag{7.5}$$

The input neurons and the bias neurons are not trained, so they are not counted towards the total training time. For small networks, this training time may be much larger than that of the weight programming training time. The area of the NMS for feature training can be estimated as:

$$\begin{aligned} \tfrac{A_{ft}}{WL} &\approx ts[3(2+N)+7(N_h+M)+3N_w \\ &+ 2(N+N_h+2)]+6(N+N_h+2). \end{aligned} \tag{7.6}$$

.

Comparing (7.6) and (7.3), we see that extra terms have been added to account for the training circuitry required by the feature training method.

7.2.3 Results for Classification

Both of the proposed off-chip training methods were tested for MNIST classification. The simulation setup is identical to the MNIST classification in the last chapter. However, instead of using random hidden-layer weights, the NMS in this section uses a regular single hidden layer MLP. An ideal model (nominal device parameters) of the NMS with 25 inputs, 20 hidden neurons and 10 outputs was trained in MATLAB using the resilient backpropagation algorithm. The results are shown in Figure 7.5(a). After 500 training epochs, the training and test classification accuracies (percentage of correctly-classified input vectors) were $\approx$94$\%$ and $\approx$85$\%$, respectively. Therefore, 85$\%$ test classification accuracy was our target and an upper limit for each of the off-chip training methods.

Behavioral models of the NMS hardware with process variations were trained using the results from the ideal NMS. The two training algorithms–weight programming and feature training–were presented in the last section. The simulation parameters for the feature training algorithm are N_{epochs}=30 and lr=2000. The value of lr is calculated from (7.4) with V_{DD}+V_{SS}=1.1 V, T_{clk}=10 μs, $I_{max} = I_c$=100 nA, and α=0.01. All of the same parameters

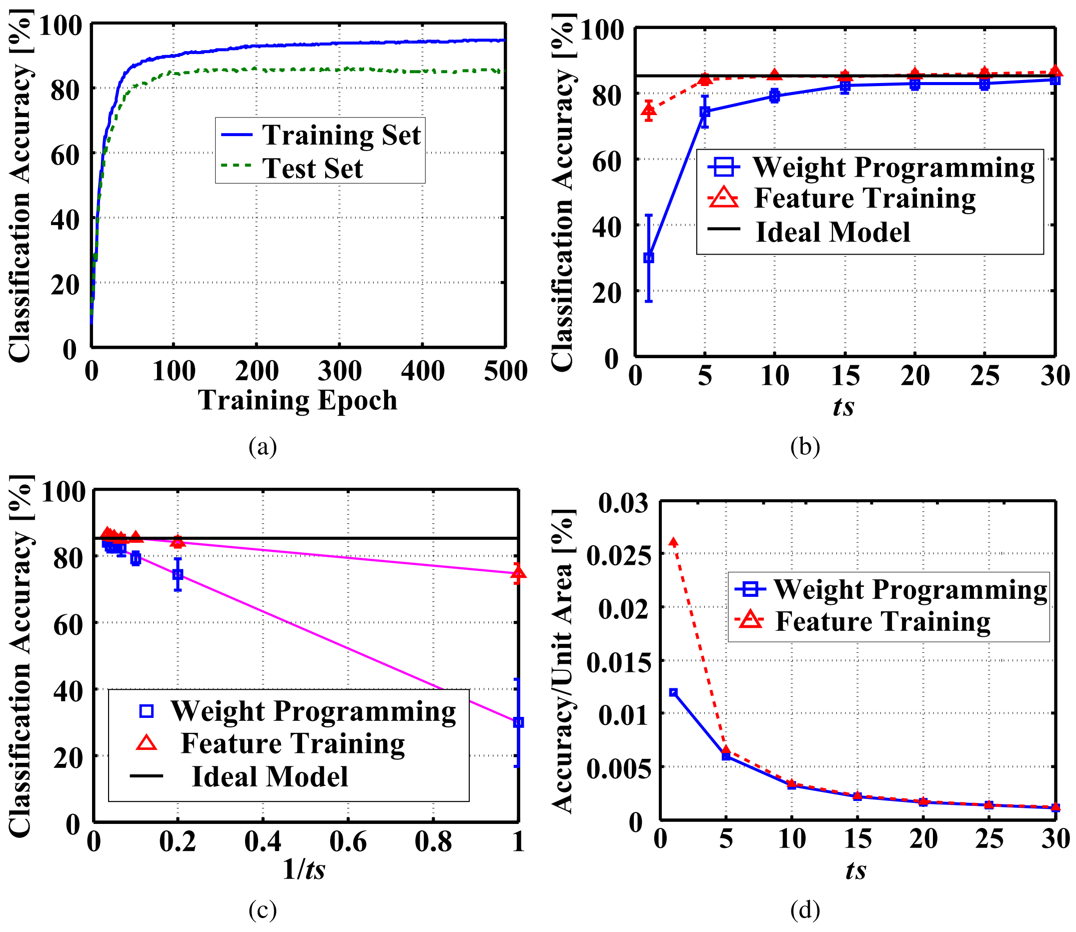


Figure 7.5: MNIST classification results: (a) Classification accuracy versus training epoch for the ideal NMS model (off-chip). (b) Classification accuracy versus the transistor size factor ts for weight programming and feature training. (c) Classification accuracy versus $1/ts$, showing the inverse area dependence of the accuracy for both methods. (d) Accuracy per unit area versus ts for both training methods.

are used for the weight programming algorithm where applicable, except for N_{epochs} which has a value of 1.

Figure 7.5(b) shows the test classification accuracies for both methods versus the transistor size factor ts. A black horizontal line is placed at 85%, indicating the accuracy of the ideal network. Each data point is the average over 10 independent runs, and error bars indicate the standard deviations. We notice a few key points from the plot. First, except for the case where ts=‘1’, the accuracy of the feature training method is relatively constant and does not vary much with ts. This indicates that, since part of the training is being performed on-chip, the NMS is learning to compensate for its own process variations. Notice that very high accuracy can be achieved with relatively low ts values. This is not the case for the weight programming method, which can not compensate for process variations. Second, we observe that even with very large ts, the weight programming method is not quite able to achieve 85% accuracy. This is likely because of the fact that not all process variations are reduced by increasing the transistor sizes. For example, the resistor R_{in} at the input of the sigmoid neuron will vary independently of ts. Also note that ts governs the transistor sizes, and the transistor areas are inversely proportional to the magnitude of the process variations. Therefore, we would expect the classification accuracy to also be inversely proportional to the $1/ts$. Figure 7.5(c), shows the classification accuracy versus 1/ts. Straight line fits show that the accuracy is in fact inversely related to the transistor area. The slope of the lines indicates the sensitivity of the accuracy to variations in 1/ts. Note that the slope for the feature training method is much smaller than that of the weight programming algorithm.

So far, the results in this section indicate that the feature training method has superior accuracy over weight programming, especially when small transistors are used. However, one must also consider the overheads associated with each method, in terms of area and performance. In Figure 7.5(d), we show the accuracy per unit area versus ts for both training methods. The unit area is the area divided by the minimum area WL, and is found using (7.3) and (7.6). For small ts, the accuracy per unit area is much larger (over 2x at ts=1) for the feature training method. However, as ts increases, the returns quickly diminish. We have also calculated the approximate training time for both algorithms using

(7.2) and (7.5). While weight programming takes only ≈ 5.5 ms, feature training takes close to 5 hours. This time can be reduced by tweaking the circuit parameters, such as increasing the memristor write voltage of the training circuit. It could also be reduced significantly by using smaller training sets. Regardless, feature training is the best method for applications where an NMS needs to be trained infrequently, and there are large area and accuracy constraints.

7.3 Summary

This chapter studied the effects of device-level variations on system-level NMS performance. On-chip, off-chip, and partial off-chip training methods were studied. The significant outcomes/conclusions are:

- On-chip training is very robust to device-level variations, but is limited to simple training algorithms.
- Off-chip training enables the use of sophisticated training algorithms, but must be implemented carefully to avoid loss of accuracy. This work proposes an off-chip training method where partial on-chip training is used to improve accuracy.

Chapter 8

Conclusions and Future Work

This dissertation has addressed several open problems related to circuit and architecture design for NMSs. Memristive circuit designs were based on semi-empirical device models to capture realistic effects observed in experimental data. To successfully move forward and design better NMSs, it is critical that device engineers work more closely with circuit designers. Currently, there is not enough breadth or depth of experimental results for any one RRAM memristor device to develop accurate models. In addition, models that are developed by device engineers are often only valid under very restricted operating conditions. It is imperative that certain device properties, such as the *on* resistance, and the number of achievable resistance states be improved. In this work, memristors were programmed by applying constant or varying-width voltage pulses. However, there are several devices that exhibit incremental resistance switching based on altering voltage pulse amplitude or compliance current. Future work should explore circuit designs that can leverage these devices as well.

At the circuit design level, this work provides new designs for synaptic communication, neuronal computation, and plasticity with reduced area and power consumption over previous work. Models were developed to describe the effects of process variations. Moving forward, it is important that the effects of PVT corners and parasitic R, L, and C components are also modeled. The designs presented in this research exhibit some obvious advantages over analog voltage-mode circuits (e.g. input/output range, reduced buffering, etc.) and digital circuits (area), but it is unclear at the present time whether NMSs are generally more efficient when implemented in an analog, digital, or spiking paradigm. It's likely that the optimal design choices will be application-dependent.

Novel training algorithms and hardware implementations were proposed to reduce design and area cost of existing training circuits. The algorithms are shown to converge through simulation, however, their convergence properties remain to be studied rigorously. One approach is to apply stochastic approximation methods to approximate the algorithm's behavior as a continuous differential equation. Further improvement of the SLMS (both online and batch versions) could be made with reduced-cost random number generation. The current method uses linear feedback shift registers, which have a large area cost.

The utility of the circuits designed in this research was demonstrated by integrating them into NMSs for visual information processing tasks. Neural network topologies were designed for edge detection, image clustering, and image classification. Performance on these tasks are modest in terms of accuracy, when compared to state-of-the-art algorithms which do not have the same constraints placed on the designs in this work (e.g. power and area). A major hurdle at this stage of research is the lack of metrics that can be used to compare designs across power, area, accuracy, and other dimensions. Overall, the results of this dissertation suggest current-mode NMSs trained with stochastic learning algorithms represent a feasible approach to designing intelligent and efficient computing architectures. In the near-term, NMSs are likely to serve as accelerators for special processes (e.g. anomaly detection) alongside conventional architectures.

Appendix A

Derivation of s_i for Constant Current Mirror Synapses

From (4.2) the post-synaptic input current is

$$i_{s_i} = \sum_{j=1}^{N_e} i_{x_j} w_{i,j} \frac{\Lambda_p\left(|v_{ds2}|, L_2\right)_{i,j}}{\Lambda_p\left(|v_{ds1}|, L_1\right)_{i,j}} + \sum_{j=N_e+1}^{N_e+N_i} i_{x_j} w_{i,j} \frac{\Lambda_n\left(|v_{ds2}|, L_2\right)_{i,j}}{\Lambda_n\left(|v_{ds1}|, L_1\right)_{i,j}} \tag{A.1}$$

Let us now focus on the Λ functions of the output transistors:

$$\Lambda_{p(n)}\left(|v_{ds2}|, L_2\right)_{i,j} = \exp\left(\Theta_{p(n)}\left(L_{2_{i,j}}\right)|v_{ds2_{i,j}}|\right). \tag{A.2}$$

For excitatory synapses, $|v_{ds2_{i,j}}| = V_{DD} - i_s R_{in}$. Substituting this into (A.2) and expanding into a second order Taylor polynomial around $i_{s_i} = 0$ leads to

$$\Lambda_p\left(|v_{ds2}|, L_2\right)_{i,j} \approx \Lambda_p\left(V_{DD}, L_{2_{i,j}}\right)\left[1 - \Theta_p\left(L_{2_{i,j}}\right) R_{in} i_{s_i} + \frac{\Theta_p^2\left(L_{2_{i,j}}\right) R_{in}^2 i_{s_i}^2}{2}\right]. \tag{A.3}$$

Similarly, since for inhibitory synapses, $|v_{ds2_{i,j}}| = V_{SS} + i_s R_{in}$,

$$\Lambda_n\left(|v_{ds2}|, L_2\right)_{i,j} \approx \Lambda_n\left(V_{SS}, L_{2_{i,j}}\right)\left[1 + \Theta_n\left(L_{2_{i,j}}\right) R_{in} i_{s_i} + \frac{\Theta_n^2\left(L_{2_{i,j}}\right) R_{in}^2 i_{s_i}^2}{2}\right]. \tag{A.4}$$

Now, plugging (A.3) and (A.4) into (A.1) leads to a quadratic equation in i_{s_i}, with the solution

$$i_{s_i} \approx \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{A.5}$$

where

$$\begin{aligned}
A &= \sum_{j=1}^{N_e} i_{x_j} w_{i,j} \frac{\Lambda_p\big(V_{DD}, L2_{i,j}\big)\Theta_p^2\big(L2_{i,j}\big)R_{in}^2}{2\Lambda_p(|v_{ds1}|, L_1)_{i,j}} + \sum_{j=N_e+1}^{N_e+N_i} i_{x_j} w_{i,j} \frac{\Lambda_n\big(V_{SS}, L2_{i,j}\big)\Theta_n^2\big(L2_{i,j}\big)R_{in}^2}{2\Lambda_n(|v_{ds1}|, L_1)_{i,j}} \\
B &= -\sum_{j=1}^{N_e} i_{x_j} w_{i,j} \frac{\Lambda_p\big(V_{DD}, L2_{i,j}\big)\Theta_p\big(L2_{i,j}\big)R_{in}}{\Lambda_p(|v_{ds1}|, L_1)_{i,j}} + \sum_{j=N_e+1}^{N_e+N_i} i_{x_j} w_{i,j} \frac{\Lambda_n\big(V_{SS}, L2_{i,j}\big)\Theta_n\big(L2_{i,j}\big)R_{in}}{\Lambda_n(|v_{ds1}|, L_1)_{i,j}} - 1 \\
C &= \sum_{j=1}^{N_e} i_{x_j} w_{i,j} \frac{\Lambda_p\big(V_{DD}, L2_{i,j}\big)}{\Lambda_p(|v_{ds1}|, L_1)_{i,j}} + \sum_{j=N_e+1}^{N_e+N_i} i_{x_j} w_{i,j} \frac{\Lambda_n\big(V_{SS}, L2_{i,j}\big)}{\Lambda_n(|v_{ds1}|, L_1)_{i,j}}
\end{aligned} \tag{A.6}$$

When channel lengths are large or the input resistances are small, $B \approx \sqrt{B^2 - 4AC}$, resulting in a loss of precision from (A.5). Therefore, we use the fact that the product of the two roots is equal to C/A, resulting in:

$$s_i \approx \frac{2C}{I_{max}\left(-B + \sqrt{B^2 - 4AC}\right)}. \tag{A.7}$$

Note that this equation is only valid when $R_{in} \neq 0$. Otherwise

$$s_i \approx \frac{C}{I_{max}} \tag{A.8}$$

Bibliography

[1] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[2] http://www.darpa.mil/Our_Work/DSO/Programs/Systems_of_Neuromorphic_Adaptive_Plastic_Scalable_Electronics_(SYNAPSE).aspx.

[3] https://www.fbo.gov/index?s=opportunity&mode=form&id=91bc9e58d6fa024d55d7c0583d38fc21&tab=core&_cview=0.

[4] http://www.darpa.mil/Our_Work/DSO/Programs/Physical_Intelligence.aspx.

[5] http://www.darpa.mil/Our_Work/MTO/Programs/Unconventional_Processing_of_Signals_for_Intelligent_Data_Exploitation_(UPSIDE).aspx.

[6] https://www.humanbrainproject.eu/.

[7] http://bluebrain.epfl.ch/.

[8] ITRS, "International technology roadmap for semiconductors," 2013. [Online]. Available: www.itrs.net

[9] L. Chua, "Memristor–The missing circuit element," *IEEE Transactions on Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, 1971.

[10] L. Chua and S.-M. Kang, "Memristive Devices and Systems," vol. 64, no. 2, 1976.

[11] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, no. 4, pp. 765–783, Jan. 2011.

[12] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing." *Nature nanotechnology*, vol. 8, no. 1, pp. 13–24, Jan. 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23269430

[13] D. Kuzum, S. Yu, and H.-S. P. Wong, "Synaptic electronics: materials, devices and applications." *Nanotechnology*, vol. 24, no. 38, p. 382001, Sep. 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23999572

[14] T. Ishigaki, T. Kawahara, R. Takemura, K. Ono, K. Ito, H. Matsuoka, and H. Ohno, "A multi-level-cell spin-transfer torque memory with series-stacked magnetotunnel junctions," in *2010 Symposium on VLSI Technology*, Jun. 2010, pp. 47–48.

[15] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-Based Resistive Switching Memories - Nanoionic Mechanisms, Prospects, and Challenges," *Advanced Materials*, vol. 21, no. 25-26, pp. 2632–2663, Jul. 2009. [Online]. Available: http://doi.wiley.com/10.1002/adma.200900375

[16] S. D. Ha and S. Ramanathan, "Adaptive oxide electronics: A review," *Journal of Applied Physics*, vol. 110, no. 7, pp. 071 101–1, 2011. [Online]. Available: http://link.aip.org/link/JAPIAU/v110/i7/p071101/s1\&Agg=doi

[17] S. Yu, B. Lee, and H. S. P. Wong, "Metal oxide resistive switching memory," in *Functional Metal Oxide Nanostructures*, ser. Springer Series in Materials Science, J. Wu, J. Cao, W.-Q. Han, A. Janotti, and H.-C. Kim, Eds. New York, NY: Springer New York, 2012, vol. 149, pp. 303–335.

[18] Y. Yang and W. Lu, "Nanoscale resistive switching devices: mechanisms and modeling." *Nanoscale*, vol. 5, no. 21, pp. 10 076–92, Nov. 2013.

[19] D. B. Strukov and R. S. Williams, "Exponential ionic drift: fast switching and low volatility of thin-film memristors," *Applied Physics A*, vol. 94, no. 3, pp. 515–519, Nov. 2008.

[20] J. J. Yang, M. D. Pickett, X. Li, D. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices." *Nature Nanotechnology*, vol. 3, no. 7, pp. 429–33, Jul. 2008.

[21] D. B. Strukov, J. L. Borghetti, and R. S. Williams, "Coupled ionic and electronic transport model of thin-film semiconductor memristive behavior," *Small*, vol. 5, no. 9, pp. 1058–63, May 2009.

[22] N. R. Mcdonald, "Al/CuxO/Cu memristive devices : fabrication, characterization, and modeling," Master's Thesis, SUNY Albany, 2012.

[23] Z. Biolek, D. Biolek, and V. Biolková, "SPICE Model of Memristor with Nonlinear Dopant Drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.

[24] A. Rak and G. Cserey, "Macromodeling of the Memristor in SPICE," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 4, pp. 632–636, Apr. 2010.

[25] Y. Zhang, X. Zhang, and J. Yu, "Approximated SPICE model for memristor," *2009 International Conference on Communications, Circuits and Systems*, no. 5, pp. 928–931, Jul. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5250371

[26] D. Batas and H. Fiedler, "A memristor SPICE implementation and a new approach for magnetic flux-controlled memristor modeling," *IEEE Transactions on Nanotechnology*, vol. 10, no. 2, pp. 250–255, Mar. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5373921

[27] C. Yakopcic, T. Taha, G. Subramanyam, and R. Pino, "Memristor SPICE Model and Crossbar Simulation Based on Devices with Nanosecond Switching Time," in *International Joint Conference on Neural Networks*, 2013, pp. 464–470.

[28] Y. Chen and X. Wang, "Compact Modeling and Corner Analysis of Spintronic Memristor Invited Paper," in *IEEE/ACM International Symposium on Nanoscale Architectures*, 2009, pp. 7–12.

[29] S. Shin, K. Kim, and S.-M. Kang, "Compact Models for Memristors Based on Charge-Flux Constitutive Relationships," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 4, pp. 590–598, Apr. 2010.

[30] 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. 3833–40, Sep. 2011.

[31] K.-t. Cheng and D. B. Strukov, "3D CMOS-Memristor Hybrid Circuits : Devices , Integration , Architecture , and Applications," in *ISPD*, 2012, pp. 33–40.

[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 Letters*, vol. 10, no. 4, pp. 1297–301, Apr. 2010.

[33] Perez-Carrasco, "On neuromorphic spiking architectures for asynchronous STDP memristive systems," in *International Symposium on Circuits and Systems*, 2010, pp. 77–80.

[34] A. Afifi, A. Ayatollahi, and F. Raissi, "Implementation of biologically plausible spiking neural network models on the memristor crossbar-based CMOS/nano circuits," *European Conference on Circuit Theory and Design*, pp. 563–566, Aug. 2009.

[35] I. E. Ebong and P. Mazumder, "CMOS and memristor-based neural network design for position detection," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2050–2060, Jun. 2012.

[36] M. Laiho and E. Lehtonen, "Cellular nanoscale network cell with memristors for local implication logic and synapses," in *International Symposium on Circuits and Systems*, May 2010, pp. 2051–2054.

[37] K. D. Cantley, "Artificial Neural Systems Using Memristive Synapses and Nano-Crystalline Silicon Thin-Film Transistors," Ph.D. dissertation, University of Texas at Dallas, 2011.

[38] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua, "Neural synaptic weighting with a pulse-based memristor circuit," *IEEE Transactions on Circuit Theory*, vol. 59, no. 1, pp. 148–158, 2012.

[39] ——, "Memristor Bridge Synapses," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 2061–2070, Jun. 2012.

[40] B. Liu, Y. Chen, B. Wysocki, and T. Huang, "The Circuit Realization of a Neuromorphic Computing System with Memristor-Based Synapse Design," in *ICONIP*, 2012, vol. 1, pp. 357–365.

[41] G. S. Rose, R. Pino, and Q. Wu, "A Low-Power Memristive Neuromorphic Circuit Utilizing a Global/Local Training Mechanism," in *IJCNN*, no. 1, 2011, pp. 2080–2086.

[42] A. Sinha, M. S. Kulkarni, and C. Teuscher, "Evolving nanoscale associative memories with memristors," in *IEEE International Conference on Nanotechnology*, 2011, pp. 861–864.

[43] M. Hu, H. Li, Q. Wu, G. S. Rose, and Y. Chen, "Memristor crossbar based hardware realization of BSB recall function," in *International Joint Conference on Neural Networks*, ser. IJCNN'12, Jun. 2012, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6252563

[44] M. S. Kulkarni, "Memristor-based Reservoir Computing," in *Nanoarch*, 2012, pp. 226–232.

[45] C. Xu, X. Dong, N. P. Jouppi, and Y. Xie, "Design implications of memristor-based RRAM cross-point structures," in *Design Automation and Test in Europe*, ser. DATE'11, 2011.

[46] R. Cell, M. Zangeneh, and A. Joshi, "Performance and energy models for memristor-based 1T1R RRAM cell," in *ACM Great Lakes Symposium on VLSI Design*, ser. GLSVLSI'12, 2012, pp. 9–14.

[47] D. Niu, "Low power memristor-based ReRAM design with Error Correcting Code," in *Asia and South Pacific Design Automation Conference*, ser. ASPDAC'12, Jan. 2012, pp. 79–84. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6165062

[48] D. Brenner, C. Merkel, and D. Kudithipudi, "Design-time performance evaluation of thermal management policies for SRAM and RRAM based 3D MPSoCs," in *ACM Great Lakes Symposium on VLSI Design*, ser. GLSVLSI'12, 2012, pp. 177–182.

[49] H. Kim, M. P. Sah, C. Yang, and L. O. Chua, "Memristor-based multilevel memory," in *International Workshop on Cellular Nanoscale Networks and their Applications*, ser. CNNA'10, vol. 1, no. 5, 2010, pp. 1–6.

[50] C. Merkel, "Thermal Profiling in CMOS/Memristor Hybrid Architectures," Master's Thesis, Rochester Institute of Technology, 2011.

[51] S. P. Adhikari, C. Yang, H. Kim, and L. O. Chua, "Memristor Bridge Synapse-Based Neural Network and Its Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 9, pp. 1426–1435, Sep. 2012.

[52] D. Querlioz, O. Bichler, and C. Gamrat, “Simulation of a memristor-based spiking neural network immune to device variations,” *The 2011 International Joint Conference on Neural Networks*, pp. 1775–1781, Jul. 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6033439

[53] M. Suri, D. Querlioz, O. Bichler, G. Palma, E. Vianello, D. Vuillaume, C. Gamrat, and B. Desalvo, “Bio-inspired stochastic computing using binary CBRAM synapses,” *IEEE Transactions on Electron Devices*, vol. 60, no. 7, pp. 2402–2409, 2013.

[54] J. V. Neumann, “Probabilistic logics and the synthesis of reliable organsisms from unreliable components,” in *Automata Studies*, C. E. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, pp. 43–98.

[55] B. R. Gaines, “Stochastic Computing Systems,” in *Computing Systems*, 1965.

[56] B. D. Brown and H. C. Card, “Stochastic Neural Computation I : Computational Elements,” *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.

[57] S. L. Toral, J. M. Quero, and L. G. Franquelo, “Stochastic pluse coded arithmetic,” in *International Symposium on Circuits and Systems*, ser. ISCAS’00, 2000, pp. I–599–I–602.

[58] W. Qian, M. D. Riedel, and I. Rosenberg, “Uniform approximation and Bernstein polynomials with coefficients in the unit interval,” *European Journal of Combinatorics*, vol. 32, no. 3, pp. 448–463, Apr. 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0195669810001666

[59] W. Qian and M. D. Riedel, “The Synthesis of Robust Polynomial Arithmetic with Stochastic Logic,” in *Design Automation Conference*, ser. DAC’08, 2008, pp. 648–653.

[60] W. Qian and J. Backes, “The Synthesis of Stochastic Circuits for Nanoscale Computation,” *International Journal of Nanotechnology*, vol. 1, no. December 2009, 2010.

[61] S. Sato, K. Nemoto, S. Akimoto, M. Kinjo, and K. Nakajima, “Implementation of a new neurochip using stochastic logic.” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 14, no. 5, pp. 1122–7, Jan. 2003.

[62] M. Pelgrom, A. Duinmaijer, and A. Welbers, “Matching properties of MOS transistors,” *IEEE J. Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, Oct. 1989.

[63] P. R. Kinget, “Device Mismatch and Tradeoffs in the Design of Analog Circuits,” *IEEE Journal of Solid-State Circuits*, vol. 40, no. 6, pp. 1212–1224, 2005.

[64] M. Pelgrom, H. Tuinhout, and M. Vertregt, “A designer’s view on mismatch,” in *Nyquist AD Converters, Sensor Interfaces, and Robustness*, A. H. van Roermund, A. Baschirotto, and M. Steyaert, Eds. New York, NY: Springer New York,

2013, ch. 13, pp. 245—-267. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-4587-6

[65] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W.-k. Shih, S. Sivakumar, G. Taylor, P. Vandervoorn, and K. Zawadzki, "Managing process variation in Intel's 45 nm CMOS technology," *Intel Technology Journal*, vol. 12, no. 2, pp. 93—-110, 2008.

[66] A. S. Oblea, A. Timilsina, D. Moore, and K. A. Campbell, "Silver Chalcogenide Based Memristor Devices," in *Proceedings of the IEEE*, vol. 3, 2010, pp. 4–6.

[67] J. G. Simmons, "Generalized formula for the electric tunnel effect between similar electrodes separated by a thin insulating film," *Journal of Applied Physics*, vol. 34, no. 6, p. 1793, 1963.

[68] D. J. Griffiths, *Introduction to quantum mechanics*, 2nd ed. Pearson Education, Inc., 2005.

[69] J. G. Simmons and R. R. Verderber, "New conduction and reversible memory phenomena in thin insulating films," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 301, no. 1464, pp. 77–102, 1967.

[70] N. F. Mott and R. W. Gurney, *Electronic processes in ionic crystals*. Oxford University Press, 1940.

[71] C. Yakopcic, S. Member, T. M. Taha, G. Subramanyam, S. Member, R. E. Pino, and S. Rogers, "A Memristor Device Model," vol. 32, no. 10, pp. 1436–1438, 2011.

[72] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "Analysis of a Memristor based 1T1M Crossbar Architecture," in *IJCNN*, 2011, pp. 3243–3247.

[73] Memristor data courtesy of Kris Campbell, Boise State University, and Knowm, Inc.

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

[75] M. Suri, O. Bichler, D. Querlioz, G. Palma, E. Vianello, D. Vuillaume, C. Gamrat, and B. DeSalvo, "Cbram devices as binary synapses for low-power stochastic neuromorphic systems: auditory (cochlea) and visual (retina) cognitive processing applications," in *Electron Devices Meeting (IEDM), 2012 IEEE International*. IEEE, 2012, pp. 10–3.

[76] S. Lin, L. Zhao, J. Zhang, H. Wu, Y. Wang, H. Qian, and Z. Yu, "Electrochemical simulation of filament growth and dissolution in conductive-bridging ram (cbram) with cylindrical coordinates," in *Electron Devices Meeting (IEDM), 2012 IEEE International*. IEEE, 2012, pp. 26–3.

[77] S. Yu and H. P. Wong, "Compact Modeling of Conducting-Bridge," *IEEE Transactions on Electron Devices*, vol. 58, no. 5, pp. 1352–1360, 2011.

[78] G. Palma, E. Vianello, C. Cagli, G. Molas, M. Reyboz, P. Blaise, B. D. Salvo, F. Longnos, and F. Dahmani, “Experimental investigation and empirical modeling of the set and reset kinetics of Ag-GeS 2 Conductive Bridging Memories,” in *4th IEEE International Memory Workshop*, 2012.

[79] H. Manem, G. S. Rose, X. He, and W. Wang, “Design Considerations for Variation Tolerant Multilevel CMOS / Nano Memristor Memory,” *Computer Engineering*, pp. 287–292, 2010.

[80] D. Niu, Y. Chen, C. Xu, and Y. Xie, “Impact of Process Variations on Emerging Memristor,” *Analysis*, pp. 877–882, 2010.

[81] M. Hu, H. Li, Y. Chen, X. Wang, and R. E. Pino, “Geometry variations analysis of TiO2 thin-film and spintronic memristors,” *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pp. 25–30, Jan. 2011.

[82] J. Rajendran, H. Maenm, R. Karri, and G. S. Rose, “An Approach to Tolerate Process Related Variations in Memristor-Based Applications,” *2011 24th Internatioal Conference on VLSI Design*, pp. 18–23, Jan. 2011.

[83] I. Bellido and E. Fiesler, “Do backpropagation trained neural networks have normal weight distributions?” in *International Conference on Artificial Neural Networks*, ser. ICANN’93, 1993, pp. 772–725.

[84] I. Bayraktaroglu, A. S. Ogrenci, G. Dundar, S. Balkir, and E. Alpaydin, “ANNSyS : an Analog Neural Network Synthesis System,” *Neural Networks*, vol. 12, pp. 325–338, 1999.

[85] B. Gilbert, “Translinear circuits: a proposed classification,” *Electronics Letters*, vol. 11, no. 1, p. 14, 1975. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el_19750011

[86] C. Toumazou, F. J. Lidgey, and D. G. Haigh, Eds., *Analogue IC Design: The current-mode approach*. Peter Peregrinus Ltd., 1990.

[87] M. P. Sah, C. Yang, H. Kim, T. Roska, and L. Chua, “Memristor bridge circuit for neural synaptic weighting,” *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*, vol. 2, no. 3, pp. 1–5, Aug. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6331434

[88] M. P. Sah, C. Yang, H. Kim, and L. Chua, “A voltage mode memristor bridge synaptic circuit with memristor emulators.” *Sensors (Basel, Switzerland)*, vol. 12, no. 3, pp. 3587–604, Jan. 2012. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3376599\&tool=pmcentrez\&rendertype=abstract

[89] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, “An energy-efficient memristive threshold logic circuit,” *IEEE Transactions on Computers*, vol. 61, no. 4, pp. 474–487, 2012.

[90] G.-b. Huang, Q.-y. Zhu, and C.-k. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 985–990.

[91] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, Aug. 2009.

[92] V. K. Rohatgi, *An introduction to probability theory and mathematical statistics*. John Wiley & Sons, Inc., 1976.

[93] K.-H. Kim, S. Hyun Jo, S. Gaba, and W. Lu, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," *Applied Physics Letters*, vol. 96, no. 5, p. 053106, 2010. [Online]. Available: http://link.aip.org/link/APPLAB/v96/i5/p053106/s1\&Agg=doi

[94] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58356

[95] C. Lu and B. Shi, "Circuit design of an adjustable neuron activation function and its derivative," *Electronics Letters*, vol. 36, no. 6, pp. 553–555, 2000.

[96] V. S. Babu and M. R. Biju, "Novel circuit realizations of neuron activation function and its derivative with continuously programmable characteristics and low power consumption," *International Journal of Advanced Research in Engineering and Technology*, vol. 5, no. 10, pp. 185–200, 2014.

[97] H. Hikawa, "A Digital Hardware Pulse-Mode Neuron With Piecewise Linear Activation Function," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1028–1037, 2003.

[98] M. Soltiz, C. Merkel, D. Kudithipudi, and G. S. Rose, "RRAM-based adaptive neural logic block for implementing non-linearly separable functions in a single layer," in *IEEE/ACM International Symposium on Nanoscale Architectures*, 2012, pp. 218–225.

[99] M. Soltiz, S. Member, D. Kudithipudi, C. Merkel, G. S. Rose, and R. E. Pino, "Memristor-based Neural Logic Blocks for Non-linearly Separable Functions," *IEEE Transactions on Computers*, vol. 62, no. 8, pp. 1597–1606, 2013.

[100] T. C. Carusone, D. A. Johns, and K. W. Martin, *Analog Integrated Circuit Design*, 2nd ed. John Wiley & Sons, 2012.

[101] C. Merkel, Q. Saleh, C. Donahue, and D. Kudithipudi, "Memristive Reservoir Computing Architecture for Epileptic Seizure Detection," *Procedia Computer Science*, vol. 41, pp. 249—-254, 2014.

[102] 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.

[103] J.-W. Jang, S. Park, Y.-H. Jeong, and H. Hwang, “Reram-based synaptic device for neuromorphic computing,” in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1054–1057.

[104] A. Schmid, Y. Leblebici, and D. Mlynek, “Two-Stage Charge-Based AnalogLDigital Neuron Circuit with Adjustable Weights,” in *International Joint Conference on Neural Networks*, 1999, pp. 2357–2362.

[105] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Allan M. Wylde, 1991.

[106] K.-h. Jo, C.-m. Jung, K.-s. Min, and S.-m. S. Kang, “Self-adaptive write circuit for low-power and variation-tolerant memristors,” *IEEE Transactions on Nanotechnology*, vol. 9, no. 6, pp. 675–678, 2010.

[107] D. McNeill, C. Schneider, and H. Card, “Analog CMOS neural networks based on Gilbert multipliers with in-circuit learning,” *Proceedings of 36th Midwest Symposium on Circuits and Systems*, pp. 1271–1274.

[108] G. Han and S. Edgar, “CMOS Transconductance Multipliers : A Tutorial,” *IEEE Transactions on Circuits and Systems*, vol. 45, no. 12, pp. 1550–1563, 1998.

[109] C. Merkel and D. Kudithipudi, “A Stochastic Learning Algorithm for Neuromemristive Systems,” in *System on Chip Conference*, 2014, pp. 359–364.

[110] B. Widrow, “An adaptive ”ADALINE” neuron using chemical ”Memistors”,” Stanford University, Tech. Rep., 1960.

[111] D. Lu, X.-h. Yu, X. Jin, B. Li, Q. Chen, and J. Zhu, “Neural Network Based Edge Detection for Automated Medical Diagnosis,” in *International Conference on Information and Automation*, no. June, 2011, pp. 343–348.

[112] E. R. Kandel, J. H. Schwartz, Thomas M. Jessell, S. A. Siegelbaum, and A. J. Hudspeth, *Principles of neural science*, 5th ed. McGraw Hill, 2013.

[113] M. Minsky and S. Papert, *Perceptrons - Expanded edition: An introduction to computational geometry*. MIT Press, 1987.

[114] C. Merkel, D. Kudithipudi, and N. Sereni, “Periodic Activation Functions in Memristor-based Analog Neural Networks,” in *International Joint Conference on Neural Networks*, no. x, 2013, pp. 1–7.

[115] S. Chen, C. F. N. Cowan, and P. M. Grant, “Orthogonal Least Squares Learning Algorithm,” *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.

[116] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, "The No-Prop algorithm: a new learning algorithm for multilayer neural networks." *Neural Networks*, vol. 37, pp. 182–188, Jan. 2013.

[117] X. He and P. Niyogi, "Locality preserving projections," ser. Advances in Neural Information Processing Systems, 2003.

[118] http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

[119] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of O(N) complexity," in *Advances in Neural Information Processing Systems*, 1988, pp. 703–711.

[120] "PJM." [Online]. Available: www.pjm.com

[121] C. Merkel and D. Kudithipudi, "Comparison of off-chip training methods for neuromemristive systems," in *International Conference on VLSI Design*, 2015.

[122] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training." *Nature communications*, vol. 4, no. May, p. 2072, Jun. 2013.